CryptoServer

Administration Guide

Utimaco Safeware AG
Transaction Security

www.utimaco.com

Imprint

Utimaco Safeware AG
Transaction Security
Germanusstraße 4
52080 Aachen

Phone ++49 (0)241 / 1696-200

Telefax ++49 (0)241 / 1696-222

Internet www.utimaco.de

E-Mail hsm@aachen.utimaco.de

Document Number 2002-0021

Version 1.2.14

Status released

Date 17th July 2008

Authors Dr. rer. nat. Gesa Ott
Dipl. Ing. Sven Kaltschmidt
Dipl. Inf. Rainer Herbertz

Table of Contents

1 Introduction

This document gives comprehensive guidelines on the administration of Utimaco's hardware security module **CryptoServer**.

The CryptoServer consists of a small computer unit which is mounted on a PCI carrier card. This computer unit is the "heart" of the hardware security module: to protect it against attacks, e. g. hostile attempts to read data out, it is encapsulated by metal shells, a special tamper detection foil and potting material. The CryptoServer's physical interface is given over the PCI interface and two serial interfaces.

Depending on the special application, the CryptoServer also has a well-defined external software interface. The CryptoServer's software concept is modular: the CryptoServer's firmware consists of encapsulated software parts, called *firmware modules*, each of them having a well-defined external and/or internal interface. Some of these firmware modules belong to the standard software of the CryptoServer, such as the operating system module, the module for basic administration functions, the module for random number generation, the module for DES cryptography, a. s. f.. Other firmware modules depend on customer's special application. Single firmware modules can be loaded, replaced or deleted by the customer after a special form of authentication. This authentication protects the CryptoServer against non-authorized access.

Main task of the **CryptoServer's administrator** is the management of the CryptoServer's firmware modules: e. g. module preparation and download, replacement and deletion of firmware modules.

*Main tool for this and other administrative tasks is the **Initialization Key**, a cryptographic RSA key which is stored e. g. on smart card or floppy. This key is used to authenticate most security-relevant administrative commands. The customer alone bears responsibility for the Initialization Key!*

Usually, if the concerned CryptoServer system is not a test system, Utimaco does not know the *Initialization Key*; in particular, Utimaco does not have a back-up copy of the key.

Because of the importance and the power of the *Initialization Key*, it is highly recommended to generate a back-up copy of the key and to store the key(s) very carefully, e. g. in a safe. Only few authorized persons, among these the CryptoServer's administrator(s), should be given access to it.

Apart from the *Initialization Key* (e. g. stored on smart card or floppy), for any CryptoServer's administration tasks the following is needed:

- a **PIN-Pad** with integrated smart card reader,
- either the **CSADM tool** (a command line utility provided by Utimaco) which must run on the PC communicating with the CryptoServer,
- or, if the CryptoServer is used in the CryptoServer LAN version, the **display menu of the CryptoServer LAN**.

This document provides a comprehensive survey on the complete CryptoServer system - hardware, software and its security architecture – while giving guidelines on its administration, including detailed description of all executable commands of the

CryptoServer. Furthermore instructions will be given about what to do in certain typical situations. It follows a rough overview of the various chapters:

- In chapter 2, *Fundamentals*, the CryptoServer hardware, its software concept and communication principles are described. This chapter is e. g. necessary for a deeper understanding of the CryptoServer's command mechanisms.
- In chapter 3, *Security Management*, the CryptoServer's life cycle and its security concept will be described. Here you will find information about the role of the various cryptographic keys used in connection with the CryptoServer and its *global states*, the responsibilities tied to these keys, and the concept for secure firmware download and firmware management. The automatic processing in case of an alarm will be described.
- In chapter 4 the usage of the CryptoServer Administration Tool CSADM will be explained. Here you will find information about its installation and the syntax of this command line utility. Detailed description of every single command and its execution is given in chapter 4.3-4.13, with the intention to help with the practical command execution and to give important tips and tricks and warnings respectively where necessary, but also a deeper understanding of each command.
- How to check the state of the CryptoServer's batteries will be explained in chapter 5. This is vital because in case that the carrier battery is not replaced early enough, all data inside the CryptoServer will be deleted.
- Chapter 6 gives practical guidance through typical administrative tasks, such as CryptoServer's first initialization or re-initialization, preparation of firmware modules for downloading or download/update of firmware modules, generation of a new *Initialization Key* and change of the *Initialization Key*. This chapter can also be used directly for help, even if you have not read chapters 2 and 3 before. You will find there direct links to chapter 4, the command descriptions.
- Chapter 7 gives help and practical guidance in critical situations like alarm, or in case the CryptoServer is not showing the expected reaction, or no reaction at all. This chapter can also be used directly, without having studied the other more comprehensive chapters before.

In order to get a real understanding of the CryptoServer system and an overview of its administration and security concept, it is recommended to read the whole document first. The detailed command descriptions in chapter 4 should be used for the execution of commands in practice.

2 Fundamentals

In this chapter the fundamentals of the hardware security module CryptoServer (and its environment) will be described:

- design principles of the hardware
- the architecture of the software
- command mechanisms and basics of the command interface.

2.1 Hardware

In this chapter the hardware of the CryptoServer will be described on a high level. All logical components will be described as far as necessary to understand where and how security has influence.

Basically the hardware of the CryptoServer consists of three components:

- a carrier card with PCI interface,
- two serial interfaces (V24) and
- the encapsulated, protected security module CryptoServer.

The connection between both modules (the PCI carrier card and the real security module) is realized over three ribbon cables.

All security-relevant components of the CryptoServer are located on the real, protected security module. The hardware of this security module, which is located on a printed circuit board and encapsulated by metal shells, a special tamper detection envelope (foil = flexible printed circuit with a serpentine geometric pattern of conductors) and potting material, is logically designed as follows:

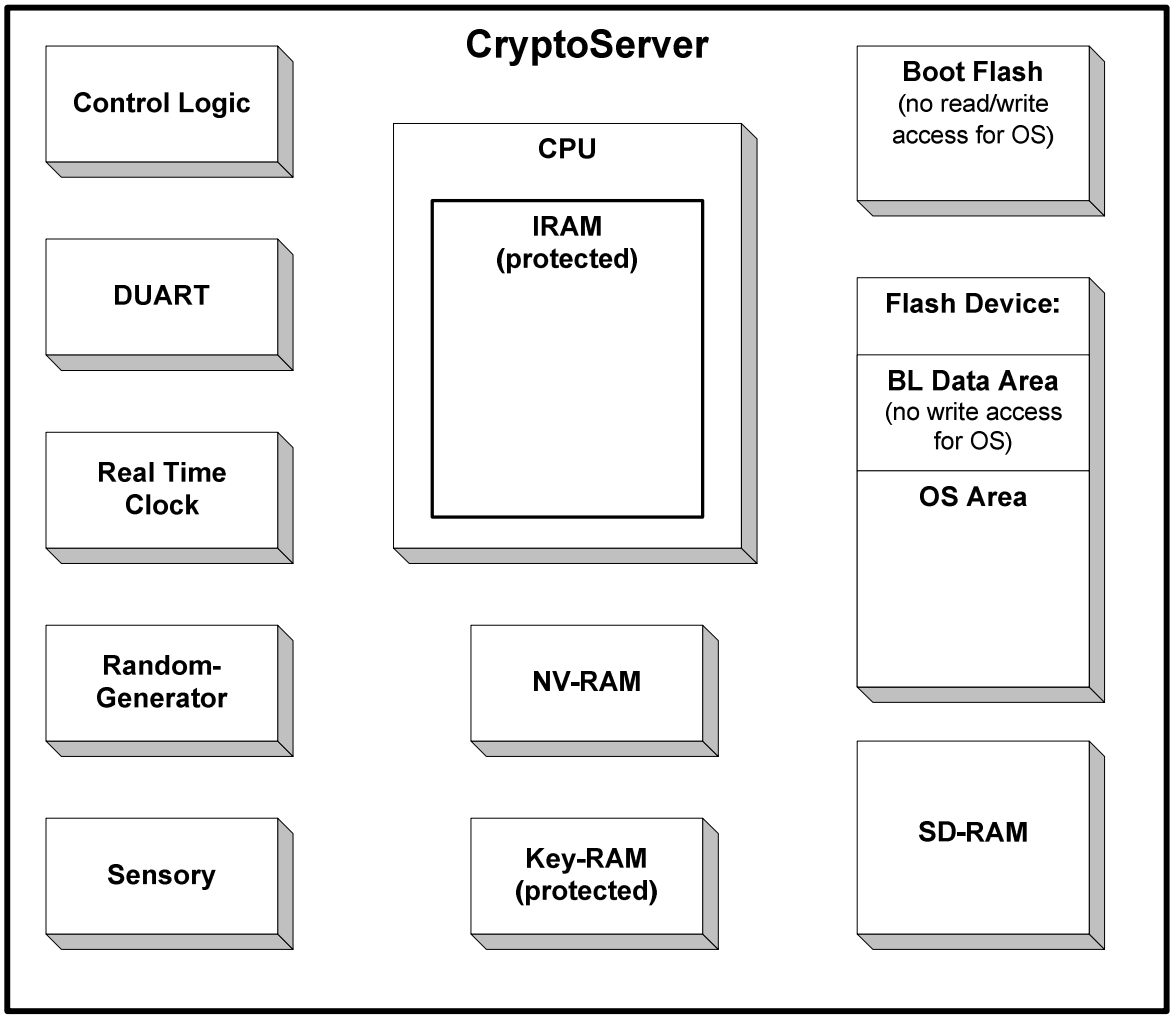


Illustration 2-1: Block Diagram of Hardware

The components of the system will be described in the following subchapters.

2.1.1 CPU

The CPU is needed to run the firmware code of the system. Cryptographic algorithms will be implemented in software and run on the CPU.

Additionally, the CPU includes memory (*IRAM*, see below). In case of an attack, the IRAM will be actively erased within a very short time.

2.1.2 Control Logic

The *control logic* will be realized in a hardware device that can be programmed only once. It is responsible for the access to several memory devices and areas.

Depending on the boot state the control logic restricts the read and write access to several memory areas in the flash file (see 2.1.7.2).

2.1.3 Hardware Random Number Generator

The *hardware random number generator* which is needed to produce real random numbers is implemented as hardware noise generator. The hardware will be watched by software: several tests will be done to control the quality of the ran¬dom bit stream, and a mathematical post-processing will be performed. Noise generator, mathematical post-processing and software tests are designed to make the true random number generator a TRNG of class P2 in the sense of the BSI standard AIS 31 [AIS31].

To get additionally the possibility of high performant random number generation the hardware generator is enhanced by a *software random number generator*. This pseudo random number generator (PRNG) is implemented according to FIPS 186-2, Appendix 3, and complies to ANSI X9.31, Appendix A.2.4. The PRNG is seeded by the TRNG and uses the SHA-1 hash algorithm as internal transition function, thus making the pseudo random number generator a DRNG of class K4 in the sense of the BSI standard AIS 20 [AIS20].

2.1.4 DUART – Serial Interface

Two serial interfaces are available. They are needed to run PIN pads for key entrance, smart card readers, printers for direct PIN envelope printing and every other peripheral device with serial interface. They can be used for the output during debug procedures.

2.1.5 Real Time Clock (RTC)

The *real time clock* (RTC) provides date and time (up to milliseconds) to the CryptoServer. It will be used for the entries of the alarm log, time log and the temperature log (adding the time to every entry). Furthermore it can be used in applications, e. g. to do time stamping.

2.1.6 Sensory

The sensory is needed to detect attacks against the CryptoServer. The following physical parameters will be watched:

- destruction of the foil (mechanical or chemical attacks)
- voltage under/overrun
- temperature under/overrun

For more details about CryptoServer's reaction in possible alarm situations see 3.3.

2.1.7 Memory Types

Several types of memory for different jobs are located inside the CryptoServer:

2.1.7.1 Boot Flash

The *boot flash* is a flash memory dedicated to store the *boot loader code* only. After any reset (power-up or hardware reset) the CryptoServer starts with the program code located in this boot flash device (non-volatile storage), i. e. with the boot loader code. This code manages a part of the system key handling and firmware download.

The code is loaded into the boot flash with a device programmer, before the boot flash device is soldered onto the CryptoServer's circuit board (and in particular before the CryptoServer is wrapped and tamper protected). A CRC checksum for code integrity checking is stored on the boot flash, too, and will be checked by the boot loader automatically during its boot process (see 2.3.3.1).

To avoid respectively to be able to detect any manipulation of the boot loader code at any later time, the firmware module ADM offers a function *CheckBootCode* which calculates and outputs the SHA-1-hash value over all code which is stored in the boot flash. This hash value can be compared with the SHA-1-hash which is calculated by Utimaco over the original boot loader code at an earlier time. This hash value comparison (being a boot loader integrity and authenticity check) will be executed by Utimaco for every single CryptoServer at the end of its production process.

The boot loader code can be updated exclusively via a dedicated boot loader command. This *Update* command can only be performed by the manufacturer himself since it has to be signed with the manufacturer's private Production Key, see 3.6.1.

2.1.7.2 Flash Device

The *Flash Device* is needed to store firmware and data (e. g. keys) non-volatile inside the CryptoServer during power-off. Storage inside the flash device will be done file-oriented (*flash file*). The flash device is divided into two areas containing data required in the different phases of CryptoServer's life cycle: the *Boot Loader Data Area* (directory SYS) and the *OS Area* (directory FLASH). These areas have limited access rights:

- The Boot Loader Data Area (directory SYS) contains data that are controlled by the boot loader but which must be read by other firmware parts, too (mainly system keys: Production Key, Initialization Key, Module Signature Key, see section 3.6, and the Ext*ended ID* (EID) of the CryptoServer, see 2.2).

  Additionally, this memory area will contain a copy of the files which have been loaded via the appropriate boot loader command *BLLoadFile*. This ought to be the first set of firmware modules loaded into the CryptoServer after initialization and must (and should only) contain the OS and the modules for communication and download of further firmware (i. e. the so-called *base firmware modules* SMOS, CMDS, UTIL and ADM, see subsection 2.3.2).

These back-up copies of the base firmware modules will be preserved even if the operating system module SMOS or other parts of the base firmware in the OS Area (see below) are replaced by future versions. In case of buggy or incompatible new modules a fallback to this first set can be made to reconfigure the whole system, see 2.3.3.

Only the boot loader firmware has read/write permission to this area, other software (including SMOS) has only read permission.

- The *OS Area* (directory FLASH) contains all program code and data required by the operating system and other firmware. The access to the OS Area is not restricted. Sensitive data inside the OS Area should be encrypted by the CryptoServer's local master key $\mathbf{K_{CS2}}$, see section 3.6.4. If the CryptoServer is attacked (e. g. by opening the foil), the sensory will erase the $\mathbf{K_{CS2}}$, so there is no chance to decrypt data and keys which are secured in this way.

  Databases which have been created by the database firmware module DB (which offers a simplified interface to store sensitive data permanently and encrypted under the $\mathbf{K_{CS2}}$, see also 3.6.4) are located here (or in the NV-RAM, see 2.1.7.5).

The limitation of access rights is guaranteed by the control logic and the implemented file system.

2.1.7.3 SD-RAM

The *SD-RAM* is a volatile RAM which is not protected by the sensory. It is needed to hold the executable code of the firmware (with exception of the boot loader code) during runtime. Also during runtime all non-sensitive data will be hold inside this memory. The SD-RAM is used as memory for program code, stack, heap and buffers.

If the sensory detects an attack, the SD-RAM will not be deleted actively. Therefore no clear sensitive data will be stored inside the SD-RAM.

2.1.7.4 IRAM

The CPU is equipped with internal RAM (IRAM) for code, data and cache.

The *IRAM* can be used to hold sensitive data in plain on runtime. It is guaranteed that the IRAM is erased actively within a very short time in case of an alarm triggered by the sensory (each memory cell of the IRAM will be overwritten), see subsection 3.3. If the CryptoServer goes down on power-off, the IRAM will be erased, too.

2.1.7.5 NV-RAM

The *NV-RAM* is a non-volatile RAM which is not protected by the sensory. Therefore no sensitive data should be stored here in clear. Databases which are created by the database firmware module DB (where sensitive data can be stored permanently and encrypted under the $\mathbf{K_{CS2}}$, see 3.6.4) can be located here (or in the OS Area of the flash file, see above).

2.1.7.6 Key-RAM

The non-volatile I^2C-RAM *Key-RAM* is sensory-protected on runtime and in retention (power supply is switched off): the Key-RAM is erased actively in case of an alarm which is triggered by the sensory. This is done by hardware immediately after the detection of the alarm by the sensory and before the CryptoServer being shutdown and restarted.

This Key-RAM is used to store the clear CryptoServer's local master key $\mathbf{K_{CS2}}$ (which will be frequently used), see 3.6.4. Independently from mode of operation the mecha¬nism to detect an attack is active. An internal power supply will provide in any case enough energy to erase the Key-RAM and therefore the $\mathbf{K_{CS2}}$ in case of alarm.

2.2 CryptoServer's Extended ID (EID)

Each CryptoServer (hardware) is identified by its *Extended Identifier* (**EID**). This EID is unique for each CryptoServer and contains information about the hardware, versions, the manu¬facturer and the customer. The EID is written by the boot loader during the production and initialization process and is part of the boot loader configuration file ('bl.ini').

The EID can be read out via the GetState *command, see 4.8.1.*

The EID consists of four 16 Bytes strings: a more hardware-linked **device ID** and the three strings **adm1**, **adm2** and **adm3** of the extended ID.

The device ID contains

- the hardware version,
- the eight bytes long, unique Unit Identifier UID of the CryptoServer's processor and
- the boot loader version (four bytes read from the boot loader code).

The three strings adm1, adm2, adm3 are each 16 bytes long and contain the following information:

Adm1: unique serial number of the CryptoServer which is assigned during the production by Utimaco Safeware AG (stored when the *Production Key* is loaded by the manufacturer)

Adm2: assigned by Utimaco Safeware AG during the first loading of the *Initialization Key*. Usually the customer's name is registered here.

Adm3: stored by the customer when he changes the *Initialization Key*.

This string can be freely assigned every time the Initialization Key is changed. But it is strongly recommended to enter the *name* of the actual Initialization Key here.

For an example of the EID see section 4.8.1.

2.3 Software

During the boot process and the life cycle of a CryptoServer several parts of its software come into action. This chapter only lists these different software components and gives a rough description of their functionality. In the following chapters about the life cycle and the boot process these items are described in more detail and as regards their global connection.

Inside the CryptoServer different kinds of software will run to different times:

- Boot loader,
- Operating system (SMOS) with firmware modules.

The boot loader is an independent firmware part that runs only partially on the CryptoServer whereas SMOS and the other modules run on the CryptoServer during its normal operational state.

2.3.1 Boot Loader

The *boot loader* is an independent software stored in the CryptoServer's boot flash and running before the real OS and the firmware modules are started. The boot loader is the first software started inside the CryptoServer after a reboot. During the CryptoServer's production at the manufacturer's site, the boot loader is responsible to load the manufacturer's public Production Key and the customer's public Initialization Key. Later on, the OS and the basic firmware modules are loaded by the boot loader into the *OS Area* (directory *FLASH*) and a copy into the *Boot Loader Data Area* (directory *SYS*) of the flash device. A detailed description of all possible boot loader commands (which offer basic administrative functionality) is given in section 4.7.

If during the boot process the boot loader finds itself *initialized* (i. e. the CryptoServer's public Initialization Key has been found) and the operating system module SMOS is present in the CryptoServer, the latter will be started by the boot loader. A more detailed description of the boot process follows in section 2.3.3.

2.3.2 Firmware Modules

This chapter will be about the firmware normally running – SMOS and the firmware modules.

Software module *or* firmware module *in the context of this documentation denotes an encapsulated software part running on the CryptoServer. A module can have an external interface which can be used by an application from outside the CryptoServer device, and an internal C interface which can be called by other firmware modules.*

The following illustration gives a general overview of the various software parts (including the software running on the host PC), how they are divided into modules and – in a rough way – where the communication paths pass.

CryptoServer Software Architecture

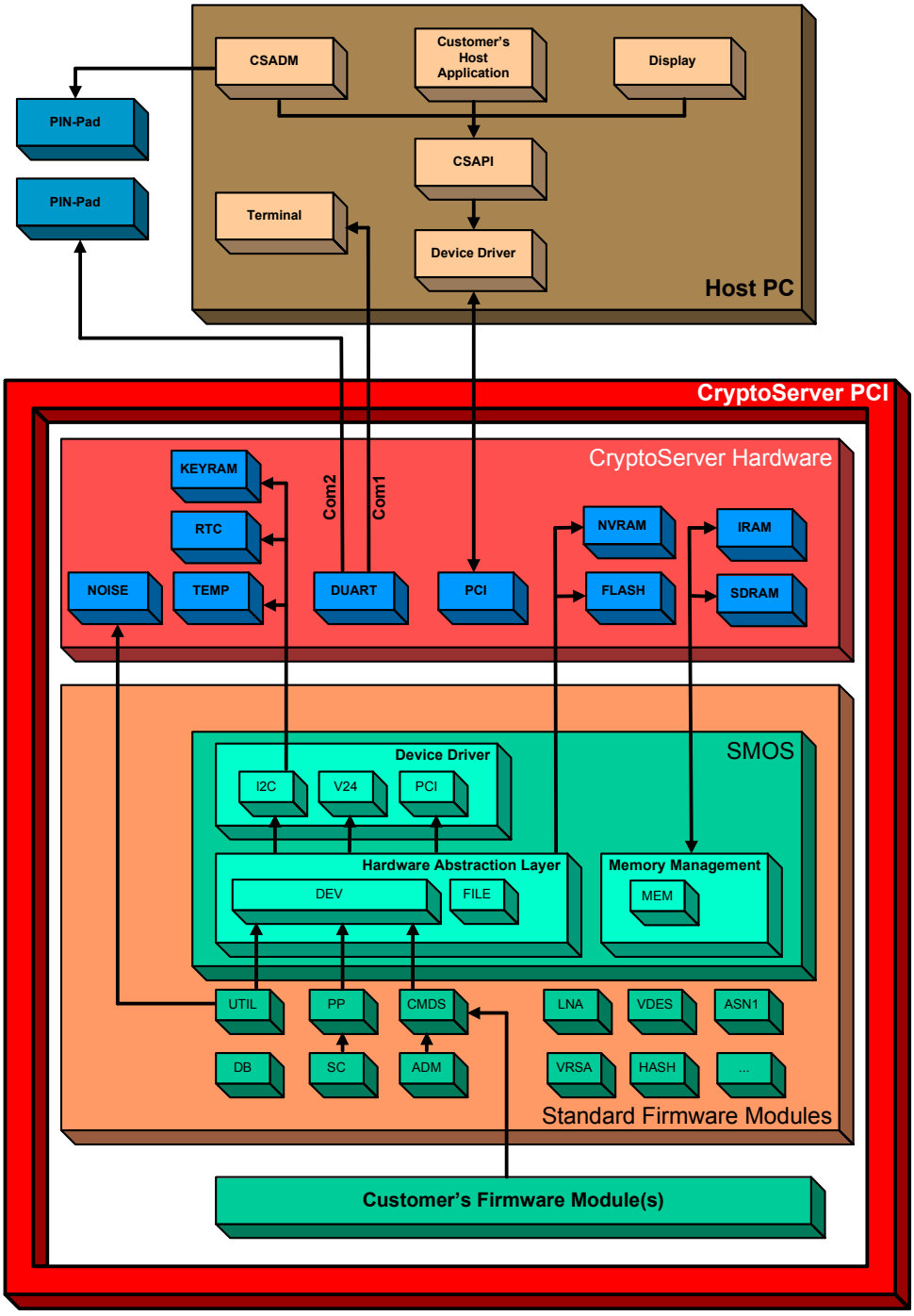


Illustration 2-2: Overview of the CryptoServer software architecture

In the above illustration, which gives an overview of the CryptoServer's software architecture in operational mode (including the software on the host PC), the abbreviations stand for the following:

- Software running on Host PC:
  - CSADM: CryptoServer's Administration Tool
  - CSAPI: CryptoServer's Application Programming Interface
- Hardware devices within the CryptoServer:
  - RTC: real time clock
  - TEMP: temperature measuring device
  - NOISE: physical noise generator as basis for the real random number generator
  - KEYRAM, NVRAM, SDRAM, IRAM, FLASH: memory devices, see 2.1.7
- Modular parts (sections) of the operating system SMOS:
  - DEV: device handling, offers generic interface for hardware access (I^2C for e. g. RTC and Key RAM, serial lines, PCI)
  - FILE: provides file system management (for NV-RAM and flash file)
  - MEM: provides memory management

For the standard firmware modules UTIL, DB, PP, ... see below. For greater clarity, most of the manifold connections and dependencies between the standard firmware modules are suppressed in this illustration. Customer application-based firmware modules have to use the CMDS firmware module (Command Scheduler, see below 2.3.2.2and 2.4.2) for the realization of their external interface but they may use all functionalities that are CryptoServer-internally provided by the other standard firmware modules, as well.

The firmware modules (running inside the CryptoServer if it is in operational mode) can be divided into several classes:

- operating system (SMOS)
- further standard firmware modules (CMDS, UTIL, ADM, VDES, ...) provided by Utimaco (which are realized in order to get a running CryptoServer with basic functionality e. g. for administration, cryptographic processes and data management) and
- other application-based firmware modules (customer's firmware modules).

Their role will be explained in the following subsections.

2.3.2.1 Operating System – SMOS

The CryptoServer's operating system SMOS (**S**mall **M**ultitasking **O**perating **S**ystem) provides mechanisms for task handling, inter process communication, memory management and file handling.

Furthermore SMOS realizes access to the several hardware components like memory areas, RTC, physical interfaces etc. and offers appropriate access command interfaces to the other firmware modules. In particular based on the included device drivers SMOS provides through its *hardware abstraction layer* high level access to the physical interfaces of the CryptoServer device (PCI and V.24). According to the normal procedure the SMOS hardware abstraction layer offers a standard set of functions (*open, I/O-control, write, read, close*) for read and write operations on the devices.

2.3.2.2 Further Standard Firmware Modules

The following modules are implemented as CryptoServer's standard firmware modules, providing the above described functionality. With only a few exceptions (e. g. the command scheduler module CMDS and the administration module ADM), these modules have no external interface and can only be accessed by other firmware modules via internal public C functions (see 2.4.1).

Name	Functionality
SMOS (basic)	Small Multitasking Operating System; operating system of the CryptoServer, responsible for receiving and sending byte streams via the physical interfaces of the CryptoServer, task scheduling, access to the CryptoServer hardware and memory
UTIL (basic)	Utilities; provides RTC access and random number generation (using the CryptoServer's hardware noise generator and a pseudo random number generator) to other firmware modules
CMDS (basic)	Command Scheduler; external interface: administration of user database (user name, given permission status, authentication method) internal functionality: responsible for distribution of received commands to the appropriate firmware modules, checks authentication of received commands CMDS is needed by all firmware modules that want to provide an external interface
ADM (basic)	Administration external interface for all maintenance jobs like loading, replacing and deleting modules, gives RTC access and CryptoServer's status and other information (e. g. name and version of loaded firmware modules)

Name	Functionality
VDES	DES; provides ■ DES algorithms (modes are ECB, CBC and CFB with 1 bit feedback) each in single and triple DES ■ DES key generation (8, 16 or 24 bytes) ■ MAC algorithms based on DES, i. e. ANSI X9.9, ANSI X9.17 (retail MAC) ■ ISO-Hash MDC-2 based on DES (ISO 10118-2) optional: ■ other DES modes like OFB or CFB with 64 or 8 bit feedback, ■ customer-defined SBOXes
AES	AES; provides ■ AES encryption and decryption in ECB or CBC mode, ■ AES key generation (128, 192 or 256 bits) and ■ AES MAC algorithm.
VRSA	RSA; provides RSA algorithm in basic mode and short-cut algorithm (Chinese Remainder Theorem), RSA key generation (variable key length)
LNA	Long Number Arithmetic; arithmetic operations with long numbers and generation of primes for RSA keys; will e. g. be used by the VRSA module for the key generation
HASH	Hash algorithms (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, RIPEMD-160)
ECA	Elliptic Curve Arithmetic; provides arithmetic operations with points on elliptic curves (which will e. g. be used by the ECDSA module for signature generation)
ECDSA	ECDSA; provides ■ ECDSA signature generation ■ ECDSA signature verification ■ ECDSA key pair generation
DB	Database; provides functions to manage data objects like keys or certificates in databases (data can be stored in clear or encrypted with the CryptoServer's local master key $\mathbf{K_{CS2}}$)

Name	Functionality
PP	PIN Pad; driver for the communication with a PIN-Pad, provides functions for text in- and output and smart card access
SC	Smart Card; driver for the communication with a smart card; provides functions for PIN authentication, changing Pins and for reading and writing data from / to protected files on a smart card. Needs PP to communicate with a PIN-Pad (smart card reader) and UTIL and VDES for secure messaging.
MBK	Master Box Key (MBK) management; management of up to four Master Box Keys (Triple DES or AES), which can be used for key back-up and restore: • MBK import (from smartcards or via PIN-Pad, according to 2-persons-rule), • MBK generation and storage (on smartcards according to 2-persons-rule), • smartcard handling If registered by any other firmware module, all MBK management functionality can also be offered as external interface. The module provides furthermore an internal interface to other firmware modules for MBK usage (for key encryption/decryption).

*The basic firmware modules (denoted with **basic** in the first column) should be loaded into the CryptoServer together with SMOS by the appropriate boot loader command* BLLoadFile*, see 4.7.6. This is necessary to get a failsafe operational CryptoServer with base functionality even in case of an emergency, able to receive further firmware.*

Basic firmware modules (operating system SMOS, CMDS, UTIL and ADM) are necessary to get the CryptoServer running with base functionality, e. g. ready for communication with the outer world, able to schedule external commands to different firmware modules and to check the authentication of the commands. Furthermore, the basic firmware modules provide administrative functions like loading, replacing and deleting further firmware modules, getting the status of the CryptoServer and reading/setting the CryptoServer-internal RTC. Additionally, they offer an internal C interface of the random number generator to other firmware modules.

All other firmware modules can and should be loaded later (in operational mode, with the appropriate ADM command LoadFile*, see 4.8.3). They run on the basis of these basic firmware modules*

The other modules provide the actual CryptoServer functionality like key storage, cryptographic algorithms or communication with a PIN-Pad.

Depending on the wanted application of the CryptoServer, possibly not all of these standard firmware modules are necessary. Thus not all of them (apart from the basic modules) must necessarily be always loaded and used.

In addition to the above mentioned standard firmware modules which are provided by Utimaco it is possible to load further firmware modules into the CryptoServer.

These so-called **application firmware modules** can e. g. offer application and/or customer dependent functionality through an appropriate external interface. Any application firmware module can use, just as any standard firmware module, the CryptoServer-internal public interfaces offered by the other (loaded) firmware modules, see subsection 2.4.1 below. The functionality and interface of application firmware is not a subject of this document but will be described in separate documents.

The functional design of each firmware module is specified in more detail in the module specific documentation (for the standard firmware modules see chapter 9).

2.3.3 CryptoServer's Boot Process

CryptoServer's boot procedure is divided into two phases each of them being controlled by one firmware part:

- first boot phase which is controlled by boot loader
- second boot phase which is controlled by the operating system SMOS

2.3.3.1 Boot Phase Controlled by Boot Loader

After any reset (power-up or hardware reset) the CryptoServer starts with the program code located in the *boot flash* device. This is the *boot loader firmware code*. The boot loader will do the first necessary start procedures. Furthermore it offers basic administration functionalities.

After a reset the CPU provided boot strap loader copies the first 1024 bytes of the boot loader code from the boot flash into the internal memory (IRAM) at address Null. As first action, the boot loader now deletes the 1 MByte internal memory of the processor and therewith all sensitive data stored there. This takes place within less than four milliseconds after a hardware reset. Afterwards the boot loader copies its rest code from the boot flash device into the internal memory of the processor and checks the integrity of the boot loader code by comparing the stored checksum (CRC) with the recalculated checksum value.

Then various self-tests and initialization steps will be performed, like

- self test (erasure and test of the SD-RAM)
- initialization and test of the real random number generator
- initialization of the flash file system
- read the boot loader's configuration file ('bl.ini')
- alarm handling (if an alarm is present)
- measurement of the temperature (power-down of the processor if it exceeds the critical limit, in this case the boot process is stopped, see section 3.4)
- determine the boot loader state of the CryptoServer (*manufactured*, *produced*, *initialized* or *defect*, depending on the result of the self test and the presence of specific keys, see section 3.2).
- initialization of the PCI interface and the serial interface and
- open time window for command (which commands are available now depends from the boot loader state, see 3.2).

If the boot loader does not receive any command within a defined time window (10 seconds), and if additionally the boot loader state is found to be *initialized* (i. e the *Initialization Key* is present, see section 3.2), if there is no alarm present and if the operating system SMOS is loaded (and can be found in either the FLASH or the SYS directory of the flash device) the boot loader will start SMOS automatically. If this succeeds the CryptoServer will be considered in the *operational* state and the boot loader will terminate itself.

If one of these conditions is not fulfilled, the boot loader will stay active and wait for further commands (depending on its state, see 3.2).

2.3.3.2 Start OS

Then CryptroServer's operating system (firmware module SMOS) can be started in two ways – either in the regular way or by starting the failsafe back-up copy of SMOS.

At the end of the boot loader-controlled phase of the boot procedure the boot loader first tries to start the OS in the usual way which means that the SMOS executable (file '*smos.mtc*') stored in the *OS Data Area* of the flash device (directory FLASH) will be copied into the SD-RAM and then started. If this fails, e. g. SMOS is not found there or is defect, the boot loader will try to start the OS with the SMOS executable stored in the *Boot Loader Data Area* (directory SYS) of the flash device (i. e. the back-up copy of the SMOS executable).

Both modalities to start the OS can also be explicitly chosen by the user if the respective external boot loader command (see 4.7.2 and 4.7.3) is performed:

- The command StartOS corresponds to the usual way of starting SMOS (i. e. from the FLASH directory) whereas
- the command RecoverOS corresponds to the SMOS-start from the SYS-directory (i. e. start of the backup copy).

If the OS start fails in both modalities the CryptoServer state will remain *initialized*. This is defined because the only reason for a failed recovery of the OS can be a missing or defect SMOS (file format error). The boot loader remains active and further boot loader commands (like e. g. loading a new file or a *Clear* to the *initialized* or *produced* state, see section 3.2.4) can be performed.

If the start of the SMOS is successful the CryptoServer goes into the global *operational* state and the boot loader terminates.

2.3.3.3 Boot Phase Controlled by Operating System SMOS

After the boot loader has passed the control to the operating system, SMOS will roughly perform the following steps:

- initialization of the memory
- initialization of the flash file system
- initialization of all hardware peripherals (timers, serial and PCI interfaces, I^2C bus)
- searching for firmware modules in the flash file system (depending on its own starting mode either in the FLASH-directory – if SMOS itself has been started from the FLASH directory - or in the SYS-directory).
- verifying the signature of all firmware modules (see 3.7.1)
- starting all firmware modules.

2.3.3.4 Watching the Boot Process and Analyzing Errors

During start-up of SMOS and other firmware modules the CryptoServer writes log messages to one of the serial interfaces, usually the *COM1 interface*. If the flash file system contains a file named "\FLASH\swap.com" then the messages are redirected to the *COM2 interface*. To watch these messages a terminal (e. g. a PC running a terminal program or the CSADM tool, see 4.15.4) has to be connected to the serial line of the CryptoServer, using the following interface settings: 115200 baud, 8 bits, no parity.

For every error or warning that occurs during the start-up of SMOS, an appropriate message is output. Additionally the log messages contain a list of all firmware modules that have been found by SMOS and whether these modules could have been started successfully or not.
If the boot process is stopped due to a fatal error, connecting a terminal to the serial line may be the only way to get information about the problem

If no fatal error occurs during the boot phase (i. e. if all basic firmware modules can be started successfully) the log messages can be retrieved later using the administration command *GetBootLog* (see 4.8.8).

2.3.3.5 Recover Back-up Copies of Firmware

If the CryptoServer hangs up itself during the boot phase, it will not be able to be accessed any more. This may happen for example if the administrator deletes one of the base firmware modules or downloads buggy software.

In such a situation of hang-up the back-up copies of the base firmware modules (which are stored in the SYS directory of the flash file, see 2.1.7.2) can be started to put the CryptoServer in operational mode again. This can be done by issuing the command ResetToBL *followed by the command* RecoverOS *(see 4.6.2 respectively 4.7.3).*

Then only the base firmware modules are running which is sufficient to administrate the CryptoServer. The bad / missing firmware can now be replaced / loaded using the normal administration command (*LoadFile* command, see 4.8.3). After that the CryptoServer can be put again in normal operational mode (running all firmware modules) with a *Restart* command.

2.3.3.6 Different Commands to Reset the CryptoServer

There are three different administration commands that reset the CryptoServer and start the boot process:

- The Reset command (see 4.6.1) causes a hardware reset. The boot process described in the previous sections is performed (including the 10 second time window of the boot loader).
- The Restart command (see 4.6.3) causes a hardware reset, waits for the beginning of the time window for boot loader commands and issues a StartOS command. The boot process is sped up by skipping the 10 second time window of the boot loader. Additionally, the Restart command waits for all firmware modules to be started by the operating system SMOS. After the Restart command has successfully finished, the CryptoServer is immediately ready to accept commands.
- The *ResetToBL* command (see 4.6.2) causes a hardware reset, waits for the beginning of the time window of the boot loader and issues a *GetState* command. The boot process is stopped and the CryptoServer remains in boot loader mode (see 2.3.4). After the *ResetToBL* command has successfully finished, the CryptoServer is immediately ready to accept boot loader commands.

If boot loader commands should be performed, use the ResetToBL *command.*
If operational commands should be performed, use the Restart *command.*

2.3.4 CryptoServer's Operator Modi: Boot Loader Mode, Operational Mode

Basically the CryptoServer can be (if the respective software is loaded) in four different modi, depending on which firmware is actually active:

*The CryptoServer is in **power down mode** if no firmware is active (CryptoServer is shutdown). In power down mode the CryptoServer is not able to receive any command. A hardware reset has to be performed to get the CryptoServer active again.*
*The CryptoServer is in **boot loader mode** if the boot loader is active, i. e. the boot loader is powered up but the CryptoServer's operating system (if loaded at all) has not yet been started.*
*The CryptoServer is in **operational mode** if its operating system as well as the base firmware modules could have been started successfully and are active so that at least basic administration of the CryptoServer can be done over these firmware modules. There are two variants possible:*

- *The CryptoServer is said to be in **normal operational mode** if it is in operational mode and the firmware modules have been started from the FLASH directory (normal way of starting).*
- *The CryptoServer is said to be in **failsafe operational mode** if it is in operational mode but the (back-up copies of the) operating system module SMOS and the base firmware modules have been started from the SYS directory. (So as long as the CryptoServer is in failsafe operational mode, no module from the FLASH directory is running).*

So, if the respective base firmware modules (SMOS, CMDS, UTIL, ADM) are loaded, the CryptoServer is in *operational mode* if at the end of the boot process the boot loader terminates and the operating system module SMOS is started, because SMOS then starts automatically the other firmware modules (if not defect).

With the *ResetToBL* command the CryptoServer can be set to boot loader mode. Certain commands can only be performed in boot loader mode. (Which commands are exactly available additionally depends on CryptoServer's *global state*, see the following subchapters).

With the *Restart* command or (if the CryptoServer is in boot loader mode) the *StartOS* / *RecoverOS* command the CryptoServer can be set to the operational mode (if the respective firmware modules are loaded and not defect). To set it to the *normal operational mode*, the *StartOS* command has to be chosen, to set it to the *failsafe operational mode*, the *RecoverOS* command has to be chosen.

With the GetState *command the operator's mode can be retrieved:*
If the CryptoServer does not answer to the GetState *command, it is in power down mode.*
In all other modi the GetState *command can be performed. A sentence*

- **CryptoServer is in Operational Mode** *or*
- **CryptoServer is in Boot Loader Mode**

(as well as the CryptoServer's state) is part of the answer to the command

2.4 Command Mechanisms

In this chapter background information will be given about how external commands are processed inside the CryptoServer and by the host PC software respectively. The difference between internal and external commands is explained beforehand. Furthermore, the mechanisms for authentication and/or secure messaging will be described. How to use these mechanisms in practice will be described later in chapter 4.

2.4.1 Public Internal Interface

*A firmware module can offer its public functions to other firmware modules as public **internal C interface**.*

The functionality provided in that way is available *only to other firmware modules inside the CryptoServer* and cannot be called from an application on the host. (Additionally, there may exist an external interface with the same functionality.)

2.4.2 External Interface

*A firmware module can offer functionality to the external world, i. e. callable from outside the CryptoServer. This command interface is called **external interface**.*

A CryptoServer external interface expects command data coded together with a command header as a **byte stream**. Such an external interface can then be attached by the *CSAPI* (or another host application) that sends/receives byte streams to/from the CryptoServer's physical interfaces (PCI interface), see below. The functions building the external interface of a firmware module will interpret the byte stream as a command like specified in the appropriate interface specification of the module. The answer of the module will be a byte stream, too, specified in the same document.

CSAPI is a software running on the host (see page 18, *Illustration 2-2: Overview of the CryptoServer software architecture*) which has the job to generate a C-interface out of the external CryptoServer byte stream interface. For this it composes the command byte stream from the different logical parts of the command header (like function code FC, sub function code SFC, command data length, ...) and the block with the specific command data and later decomposes the answer byte stream into the different logical parts of the answer header and the block with the specific answer data. This C-interface can then be used by the host application: it hides the composition and decomposition of the command/answer header byte stream from the application programmer and therefore facilitates the usage of the CryptoServer protocol stack (in particular the usage of the authentication layer). More details about the CSAPI will be given in the next chapter 2.4.3.

The task of the composition/decomposition and interpretation of the function specific command/answer data is left for the host application software. This has to be done pursuant to the appropriate module specification. For the external interface of the standard firmware modules (e. g. of the CMDS or ADM modules) this job is done by the *CryptoServer Administration Tool CSADM* (which attaches on the CSAPI, see *Illustration 2-2*). This command line utility will be described extensively in chapter 4.

Inside the CryptoServer, an external command of a module will be executed directly when it is called:

For external commands, there is only single command processing ("store and forward") available. The commands will be performed one after the other, in that order in which they are received by the CryptoServer.

Responsible for the processing of the protocol stack inside the CryptoServer is the firmware module CMDS:

First CMDS gets the input (command) byte stream from the PCI interface via the PCI driver provided by the operating system SMOS. CMDS then processes the command header and, if everything is correct, passes the pure command data to the specific function of the specific firmware module (which are announced through the FC and SFC in the header).

2.4.3 Command Processing on the Host PC: CSAPI

The **CryptoServer Application Programming 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 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.

CSAPI offers C functions to

- open or close a connection to a CryptoServer.
- push or remove specific protocol layers onto the protocol stack (see below for the possible layers).
- send and/or receive a data block of an external command to the CryptoServer. This includes the construction respectively analysis of the byte strings from/to the given C parameters, according to every protocol layer.

The specific protocol layers that can be used with the CSAPI are:

- CMDS – mandatory, always the top level layer on the protocol stack

- 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)

The protocol layer **CMDS** is a simple command format for the CryptoServer with the purpose to direct the command data to the specific function of the specific firmware module. It must be the top level layer on the protocol stack (for normal CryptoServer operations, i. e. CryptoServer is in operational mode: the operating system SMOS and the base firmware modules are running).

The protocol layer **AUTH** is used to authenticate a CryptoServer command, or to log in and logoff into the CryptoServer. Therefore AUTH uses the *user table* where for each user there is a name, his permissions and specific data for his specific authentication mechanism stored (e. g. the password or the public RSA or ECDSA key). See also chapter 2.6.

The protocol layer **CHNL** is a simple format for adding a channel number to a CryptoServer command.

The protocol layer **SM** is used for secure messaging between CryptoServer and the host. With an active SM layer every command and answer block sent to / received from the CryptoServer is encrypted and protected with a MAC. See also chapter 2.7.

The protocol layer **BL** is a simplified version of the layer CMDS that is used for communication with the boot loader of the CryptoServer. 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).

A detailed description of the CSAPI and the protocol layers can be found in [CSAPI].

2.5 User Concept

Security-relevant external commands that are sent to the CryptoServer may only be performed if the sender has authenticated *the command. Such command authentication can only be done by so-called* ***users*** *which have to be registered at the CryptoServer.*

To control and restrict the access to security-relevant commands, the CryptoServer implements the concept of **users**[1]: Authentication of commands can only be done by registered *users* who are equipped with the relevant *permissions*. Only those users are allowed to access security-relevant commands.

For implementation of the user concept, the CryptoServer administrates a *user data base*. There for each *user* the following data will be stored:

- *Name* (which serves as a unique identifier for the user), up to 255 bytes long.
- *Permissions* of the user. The structure of these user permissions corresponds to the structure of the authentication state (as explained in the section 2.6.1 below), i. e. it consists of eight values in the range from 0-3.
- *Flags* that determine if *static login* or *secure messaging* is allowed for this user.
- The *authentication mechanism* that has to be used by the user (see 2.6.3 below).
- *Authentication data* like cryptographic keys, passwords or others, see below.
- *User Attributes* (optional).

For a more detailed descrption of the possible user data see 4.8.17. The CryptoServer provides also the appropriate commands to create or delete a user or to change his/her authentication token (data), see 4.8.17.

A user with user name ADMIN*, the* CryptoServer's administrator*, is always present in the CryptoServer. ADMIN is able to perform all CryptoServer's administration, in particular the administration of firmware modules, of the RTC and of the user database.*

If the CryptoServer is in bootloader mode, the user ADMIN is the only user available. For more details about ADMIN see section 2.6.4.

After a successful authentication the CryptoServer's *authentication state* will be augmented by the permissions of the user. The authentication concept will be explained in the following chapter.

[1] Here the CryptoServer is considered to be in *operational mode*, see 2.3.4. For a CryptoServer in *boot loader mode* only one user, ADMIN, is available, see below.

2.6 Authentication

The CryptoServer accepts certain external commands only after one (or more) appropriate user(s) have been successfully authenticated.

Inside a CryptoServer, the firmware module CMDS is responsible for the distribution of received commands to the appropriate firmware modules. The CMDS module is also responsible for the authentication of commands: [2]

Certain external commands that are sent to the CryptoServer may only be executed if the sender has authenticated the command and a certain *authentication state* has been reached (see below). For this purpose one or more authentication header data blocks can be added to the command data block. These authentication headers will be processed completely by CMDS, and, depending on the result, CMDS will increment the authentication state. The addressed firmware module which will only receive the command data block is then responsible for checking the authentication state, if necessary, and to decide about its further execution.

The following subsections will explain the authentication mechanisms and usage in detail.

2.6.1 Authentication State

The CMDS firmware module stores an **authentication state** internally. The stored value will be incremented after every successfull authentication. Depending on the value of the current authentication state it will be decided if a command is allowed to be performed or not:

The authentication state consists of eight values, each in the range from 0 to 3. These eight values represent eight different **user groups** (user group 0 to 7). Each value, called **authentication level**, gives the height (sum) of the rights/permissions gained through the authentications of various users from this specific user group. Each authentication level can vary from 0 (no authentication) to 3 (highest level of permission).

Internally, this authentication state is realized through a 4 bytes integer where each nibble (half byte) represents one user group and can take a value between 0 and 3. The least significant nibble stands for the user group 0, the most significant nibble stands for the user group 7.

[2] Throughout this chapter, the CryptoServer is considered to be in *operational mode*, see 2.3.4. For a CryptoServer in *boot loader mode*, other, simplified authentication mechanisms will be applied, see 4.7.

Example:

	Eight half bytes (nibbles) of example authentication state							
Value of Authentication state:	2	0	0	1	0	3	0	1
Nibble for user group ...	7	6	5	4	3	2	1	0

In this example, the CryptoServer has reached authentication level 2 in user group 7, level 0 in user group 6, (...), and authentication level 1 in user group 0.

A successful authentication only changes this authentication state. It is up to each individual command (and thus a task of the respective firmware developer) to check the current authentication state and, depending on its value, to decide if it will continue with execution or not.

Thus the meaning of the authentication level within a specific user group has to be defined completely application dependent and during firmware development. Later, i. e. for a given implemented firmware, the relation between commands and user groups cannot be administrated.

For a given firmware/application, a specific security policy (e. g. the 2-persons-rule for sensitive commands) can only be realized over the User Management, see 2.6.4 below

2.6.2 Authentication Modes: Static Login, Single Command Authentication

For the authentication, two different **authentication modes** exist:

1. **Static Login:**
   After a *static login* (see 4.10.8) the achieved authentication state is kept until it is set back by an explicit *logoff* (see 4.10.9). During this time, all commands for which the achieved authentication state suffices can be performed without any further authentication.
2. **Single Command Authentication:**
   This means that the authentication holds only for a single command and must therefore be done together with this command (see 4.10). After the execution of the command, the authentication state will be automatically set back to the previous value.

Static login should be used very carefully, and only if it can be made sure that - during the time when a user is statically logged in - no unauthorized user is able to send commands to the CryptoServer. It is possible to forbid static login for specific users or even for all users, see 2.5.

2.6.3 Authentication Mechanisms

The following different authentication mechanisms are implemented:

1. **Cleartext Password Authentication:**
   For this mechanism a cleartext password will be passed from host to CryptoServer. CryptoServer compares this password with the one stored in the user database for the respective user. A password consists of 16 bytes.

2. **SHA-1 Hashed Password Authentication:**
   For this mechanism also a 16 bytes long password will be used. First the host demands an 8 bytes random value from the CryptoServer. Then the host calculates the SHA-1 hash value over this random value, the password and the command data block. It transfers this hash value to the CryptoServer which recalculates and checks the hash with the help of the password stored in the user database. Compared with the cleartext password authentication, this mechanism has the following advantages:

   - The password will not be submitted in clear and thus can not be eavesdropped.
   - Because of the random value the authentication data block cannot be eavesdropped and replayed at a later time.
   - The command data are protected against unnoticed manipulation.

   For these reasons, this mechanism is also qualified for the communication via Ethernet (CryptoServer LAN).

3. **HMAC Password Authentication:**
   For this mechanism a password of arbitrary length will be used. First the host demands an 8 bytes random value from the CryptoServer. Then the host calculates the HMAC value over this random value and the command data block using the password as the HMAC key. It transfers this hash value to the CryptoServer which recalculates and checks the hash with the help of the password stored in the user database. The default hash algorithm for the HMAC calculation is SHA-256. Other hash algorithms can be used on demand.

   This mechanism has the same advantages than the SHA-1 hashed password authentication and is therefore also qualified for the communication via Ethernet (CryptoServer LAN).

4. **RSA Signature Authentication:**
   For this mechanism the host demands again an 8 bytes random value from the CryptoServer first. Then the host calculates a RSA signature over this random value and the command data block with a private RSA key (PKCS#1 signature format). This signature will then be transmitted to the CryptoServer which will verify it with the help of the RSA key's public part which is stored in the user database. The default hash algorithm for the signature calculation is SHA-1. Other hash algorithms can be used on demand.

   This mechanism is particularly qualified for communication via Ethernet (CryptoServer LAN) and therefore recommended for secure applications.

5. **ECDSA Signature Authentication:**
   For this mechanism the host demands an 8 bytes random value from the CryptoServer first. Then the host calculates an ECDSA signature over this random value and the command data block with a private ECDSA key. This signature will then be transmitted to the CryptoServer which will verify it with the help of the ECDSA key's public part which is stored in the user database. The default hash algorithm for the signature calculation is SHA-256. Other hash algorithms can be used on demand.

   Just as the RSA signature authentication, this mechanism is particularly qualified for communication via Ethernet (CryptoServer LAN) and therefore recommended for secure applications.

6. **RSA Smart Card Authentication:**
   For this mechanism the entire authentication is performed within the CryptoServer, with the help of RSA signature smart cards: Over a PIN-Pad (including smart card reader), which is *directly* connected to it, the CryptoServer requests a signature and verifies it using the public part of the RSA key which is stored in the user database. This mechanism is only applicable for local applications since direct access to the CryptoServer is necessary, it is not applicable if the CryptoServer is remotely used.

The detailed structure of the authentication data blocks is described in [CSCMDS]. For the syntax which has to be used in this context with the CSADM tool, see sections 4.10.1-4.10.6.

2.6.4 Users and User Permissions

As explained in 2.5 above, commands can only be successfully authenticated by authorized **users**. For the management of these *users*, the CMDS module uses and administrates a **user database** with the data entries for every user as listed in section 2.5.

For every authentication to be performed, the user name and the authentication mode (static login or single command authentication, see above) have to be specified. Further necessary data depend from the authentication mechanism of the user.

After a successful authentication the authentication state will be augmented by the permissions of the user.

Example:

Authentication state before user's login:	01000000
Permissions of the user:	01000001
Authentication state after user's login:	02000001

Several users can authenticate/login one after the other or within one command. Doing this, mixed authentication modes (static login or single command login) and authentication mechanisms are allowed.

There are two user groups predefined for the access to the external commands of the standard firmware modules:

User groups 6 and 7 are reserved and should not be used by application firmware modules. User groups 0 to 5 are available for user defined applications

User groups 6 and 7 are reserved for the following tasks:

- For the administration of firmware modules and setting of the CryptoServer's RTC (loading/replacing/deleting firmware modules and setting the system clock, see 4.8) an authentication level of 2 for the user group 6 is mandatory.
- For the administration of the user database (create or delete users, see 4.8.17) an authentication level of 2 for the *user group 7* is mandatory.

A user with user name ADMIN is always present in the CryptoServer.

The authentication mechanism of ADMIN is 'RSA Signature Authentication' with the CryptoServer's private *Initialization Key* $\mathbf{K_{INIT\_PRV}}$ (see 3.6.2). The user ADMIN cannot be changed or deleted in the user database. The user permission of ADMIN is '22000000', i. e. authentication level 2 for user groups 6 and 7. This means that ADMIN is allowed to perform all standard administration and user management commands, see above.

The implementation of the standard firmware demands authentication level 2 in user group 6 resp. 7 for the execution of the administration and user management commands. This allows the realization of customer individual security rules by setting rules for the creation of users, e. g.:

- *If the customer's security rules shall allow execution of these commands after authentication of one authorized user, then users with permission level 2 in user groups 6 and 7 can be created.*
- *If the customer's security rules shall allow execution of these commands only after authentication of two independent authorized users (i. e. according to the 2-persons-rule), then only users with permission level 1 in user groups 6 and 7 should be created.*
- *Every other security rule which requires authentication according to the 2-persons-rule only for specific commands or specific users can be realized that way by setting respective rules for the user management.*

So the implementation of security rules for the authentication of commands within a specific Crypto¬Server application is dependend upon the application firmware as well as upon the user management:

- *With the concrete implementation of application firmware modules the frame for the security rules for command authentication will be determined. In particular it will be fixed which commands have to be authenticated and which not.*
- *Within this frame, with setting rules for user administration, the customer-individual security rules for command authentication can be realized. This can be done by setting the specific permissions and authentication mechanisms for any user.*

2.7 Secure Messaging

The CryptoServer supports 'Secure Messaging' for the communication between the CryptoServer and the host: commands sent to the CryptoServer and answer data received from the CryptoServer may be AES encrypted and integrity-protected with an AES MAC. For this purpose a secure messaging header data block can be added to the command and answer data block. The detailed structure of the header data blocks is described in [CSCMDS].

To use the secure messaging functionality, two steps are required (each of them being transparent to the user of the CSADM tool since performed within one CSADM command, see below):

1. **Generate a session key.**

   First the external CryptoServer function *GetSessionKey* is called to generate an AES session key.

   The session key can be negotiated either with the *Diffie-Hellman* key establishment protocol (see [PKCS#3]), the *Anonymous Elliptic Curve Diffie-Hellman* key establishment protocol (see [ANSI-X9.63] section 6.2) or using the secret of a user registered in the CryptoServer.

   If one of the *Diffie-Hellman* key agreements is used, no user account is necessary to generate the session key.

   In the other cases the user's secret (his key or password) will be used to encrypt or to establish the session key. Secure messaging must be allowed for the selected user (user flags). The CryptoServer also returns a session ID and a starting value of a sequence counter. The exchange of the session key is done in the following way:

   If the user uses *password authentication*, the CryptoServer will calculate the SHA-256 hash value over the user's password and the actual sequence counter. Using this 32 bytes result value as key encryption key the CryptoServer returns the session key *encrypted* to the host.

   If the user uses *RSA authentication*, the CryptoServer will encrypt the session key with the public part of the user's RSA key and send it to the host.

   If the user uses *ECDSA authentication*, the session key will be negotiated over the Elliptic Curve Diffie-Hellman key establishment protocol with the user's ECDSA key pair (according to [ANSI-X9.63] section 6.2).

2. **Send encrypted commands.**

   The host can send commands to the CryptoServer that are AES-encrypted and integrity protected with a AES MAC using the secure messaging layer (layer SM) described above in 2.4.3. The respective answers to these commands, which are sent back to the host by the CryptoServer, are always encrypted and protected with a MAC, too. The sequence counter is used as initialization vector for the AES encryption and MAC calculation and is incremented after every command to prevent unauthorized replays of the commands.

The session key used is identified by a session ID. All commands using the same session ID and the same session key are said to belong to one session. In this way a secure channel can be established between the CryptoServer and the host application using the Secure Massaging mechanism

After the CryptoServer has generated a session key, it still accepts commands in clear, i. e. without secure messaging. But if the CryptoServer receives an encrypted command (i. e. a command using layer SM), it checks the MAC. The command is rejected if the MAC is invalid.

- *A session key automatically becomes invalid if it has not been used to encrypt any command for more than 15 minutes.*
- *A maximum of 256 session keys may be active at the same time and can be used by different host applications simultaneously (each key identified by its session ID). If a host application requests a new session key while the maximum of 256 sessions are already active, the oldest session is closed and its session key is invalidated.*

If the *GetSessionKey* function is authenticated by one or more users using single command authentication, the permission of this user(s) will be granted to the whole session. All commands that are encrypted with this session key have the permission of these users without extra authentication. But outside this session the former authentication state is preserved.

To the user of the CSADM tool, the steps explained above for the usage of secure messaging remain transparent: The user just has to perform

- the *SessionRSA* command (for a user with RSA signature or RSA smart card authentication mechanism, see 4.11.1),
- the *SessionEC* command (for a user with ECDSA authentication mechanism, see 4.11.2),
- the *SessionPwd* or *SessionHMAC* command (for a user with password authentication mechanism, see 4.11.3 and 4.11.4)
- or the *SessionDH* or *SessionECDH* command (for Diffie-Hellman key agreements, usable by any user, see 4.11.5 and 4.11.6),

together in one command line with the command(s) for which secure messaging should be used. Then CSADM will automatically open the session (by getting the session key), perform the specific command(s) with secure messaging (i. e. encrypted and MAC-secured) and close the session on the CryptoServer again.

For details on the usage and syntax of secure messaging see section 4.11.

3 Security Management

In this chapter the CryptoServer's life cycle will be described and, among other things, the following questions will be answered:

- What are the possible CryptoServer's global states and how are they connected to the CryptoServer's life cycle?
- How does the CryptoServer get from one state to another?
- Who is responsible for the CryptoServer in a certain state?
- What is the role of the various cryptographic keys used in connection with the CryptoServer?
- How does secure firmware download work?

3.1 CryptoServer's Life Cycle

CryptoServer's life cycle denotes its complete lifetime, from manufacturing to the operational (running) phase until destruction. A CryptoServer will run through the phases and *global states* given in this picture and described in more detail in the following chapters. [3]

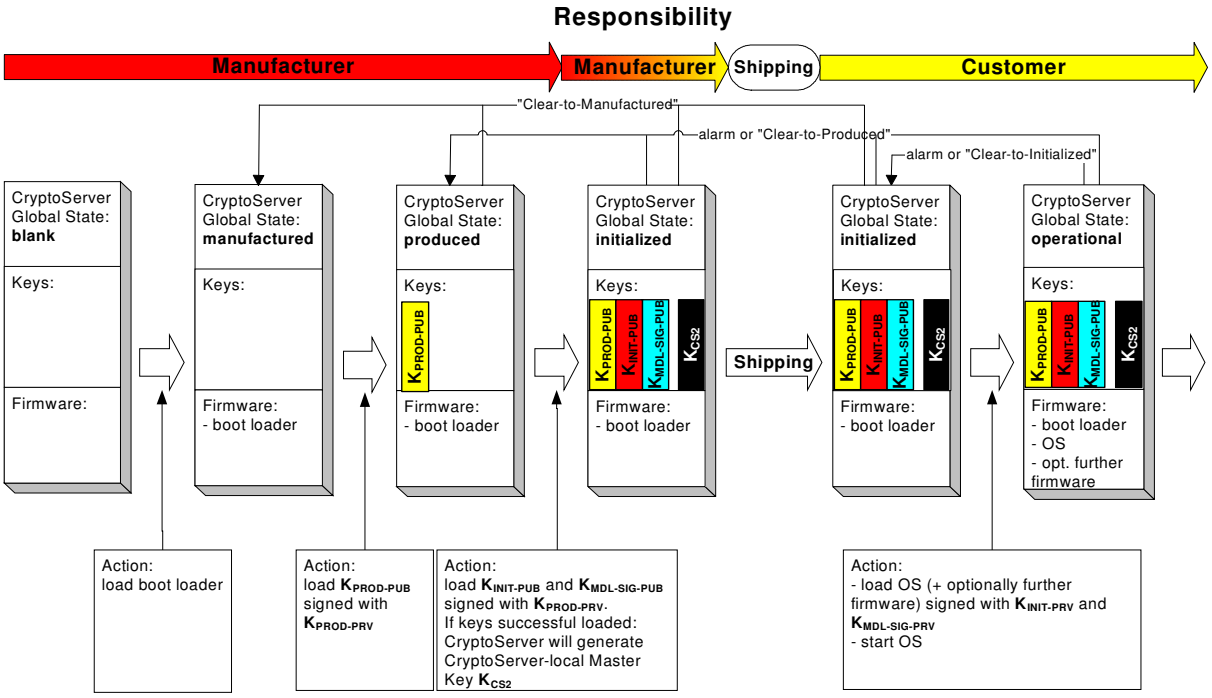


Illustration 3-1: Responsibility during the CryptoServer's life cycle

[3] Additional phases of life which are possibly added at a later date and depending on the type of application software running on a CryptoServer are not described here.

From this illustration it can be seen that CryptoServer's *global state* depends on the loaded data, in particular on the presence of specific cryptographic keys. A short description of the possible states and the respective responsibilities follows:

State	Description	Responsibility
blank	The CryptoServer is virginal, there is neither any firmware nor any key inside. There is no housing present yet	**The CryptoServer is related to no one. It is located within the manufacturer's secure environment.** The responsible person or organization is able to load the boot loader software.
manufactured	The boot loader is loaded into the CryptoServer's boot flash device. The mechanical housing of the CryptoServer is closed. The tamper foil is wrapped around the housing, the CryptoServer is potted and mounted on the PCI carrier card.	**The CryptoServer is related to no one.** The responsible person or organization is able to load the public key $\mathbf{K_{PROD\text{-}PUB}}$ which is related to the manufacturer. **In the *manufactured* state the CryptoServer is located within the manufacturer's secure environment.**
produced	The boot loader is running and the manufacturer's public *Production Key* $\mathbf{K_{PROD\text{-}PUB}}$ is loaded.	**The CryptoServer is related to the manufactorer** The responsible person or organization is able to initialize the CryptoServer (and thus to asign it to a customer). **In the *produced* state the CryptoServer will not leave the manufacturer's site.**
initialized	The boot loader is running and the customer's public *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$ is loaded. Together with $\mathbf{K_{INIT\text{-}PUB}}$ the manufacturer's public key for module signature $\mathbf{K_{MDL\text{-}SIG\text{-}PUB}}$ is loaded and, CryptoServer-internal, the local Master Key $\mathbf{K_{CS2}}$ is generated and stored in the (protected) *Key RAM*.	**The CryptoServer is related to the customer.** The responsible person or organization is able to load firmware modules.
operational	The operating system SMOS is loaded and started successfully. Optionally further firmware is present.	

State	Description	Responsibility
defect	During the boot process the basic hardware self test (which has been performed by the boot loader) detected a malfunction.	**The person/organization that has been responsible before is still responsible for the CryptoServer.** Only the commands *GetState* and *GetBootLog* are available (if technically possible).

Table 3-1: States of the CryptoServer during its life cycle

For a detailed description of the CryptoServer's states as well as the related keys and their exact meaning/role please see the respective chapters which will follow later on.

One requirement of the life cycle is to have the possibility to document it completely. For this the responsible persons have to write protocols for every (sensitive) action done with each CryptoServer. The protocols should include at least the following data:

Data	Description
Timestamp	day and time the action took place
CryptoServer-ID	identification of the security device (EID or UID, see 2.2)
Person(s)	information about the person(s) responsible for and/or participating in the action
Action	sort of action done

Table 3-2: Data in the Protocols Documenting the CryptoServer's Life Cycle

These protocols may be written by hand or may be produced by tools used during the action. In either case they must be signed by the responsible person (by hand or electronically).

3.2 Global States of the CryptoServer

In error-free operation the CryptoServer runs through the following states:

BLANK -> MANUFACTURED -> PRODUCED -> INITIALIZED -> OPERATIONAL

A fall-back to an earlier state happens either in case of alarm (damaged foil, voltage too high/¬low, temperature too high/low) or if the user changes the state on purpose:

In case of an alarm, within a very short time the Key-RAM and therefore the CryptoServer's local *Master Key* $\mathbf{K_{CS2}}$ will be erased and the CryptoServer will be restarted. As first action after the restart the boot loader erases the internal memory (IRAM) and further data such that the CryptoServer falls back to the *initialized* state after an alarm (see 3.3).

Furthermore, the user has the possibility to reset the CryptoServer on purpose to one of the previous states: thereby the clearing command *BLClear* with its various erase options (which has to be suitably authenticated, see 3.5 and 4.7.4) can be used.

At the customer's site only the CryptoServer's initialized *and* operational *states will normally occur. If the CryptoServer is found to be in any other state, the manufacturer/Utimaco Safeware AG has to be contacted.*
Global states such as blank*,* manufactured *and* produced *are exclusively relevant for the CryptoServer's production process and for maintenance work done by the manufacturer. Utimaco will never deliver the CryptoServer in one of these states.*

It follows a detailed description of the CryptoServer's possible states and the external commands which can be performed in those respective states.

3.2.1 State: Blank

In this state the CryptoServer is assembled but no boot loader or any other firmware or keys are loaded. The housing has not been done yet.

3.2.2 State: Manufactured

After loading the boot loader, finally assembling the CryptoServer (i. e. do the housing and potting) and activating the sensory mechanism, the CryptoServer will enter the *manufactured* state. From now on it does not leave the secured production environment until the public part of the manufacturer's *Production Key* $\mathbf{K_{PROD\text{-}PUB}}$ is loaded, since no command authentication is possible prior to that.

In the *manufactured* state the boot loader accepts the following commands:

Command	Description
GetState	ask for information about state, hardware and temperature
BLGetAlarmLog	ask for the alarm history
BLGetTempLog	ask for the temperature history
BLResetAlarm	reset of a previous alarm If *BLResetAlarm* is performed but the physical alarm is still present, the hardware automatically triggers a new reset and the boot loader will be started again (see also 3.3).
BLLoadProdKey	load the public part of the *Production Key* $\mathbf{K_{PROD\text{-}PUB}}$
BLClear	clear the CryptoServer to the state • *manufactured*

Since the boot loader accepts these commands without authorization, the CryptoServer will be taken into the next state (state *produced*) very soon.

The CryptoServer state manufactured *exclusively occurs at the manufacturer's site and never at the customer's site. Therefore the above mentioned commands without command authentication cannot be performed by the customer.*

3.2.3 State: Produced

After the public part of the *Production Key* $\mathbf{K_{PROD\_PUB}}$ has been loaded, the CryptoServer is in the *produced* state. It is now possible to authenticate commands with a RSA signature calculated by the private part of the manufacturer's production key $\mathbf{K_{PROD\text{-}PRV}}$, which can be verified by the boot loader with the public part of the key. Therefore the CryptoServer can leave the secure production environment. Nevertheless it will remain at the manufacturer's site until its *initialization* (*initialized* state, see next chapter). In this state, only the manufacturer can perform command authentication because only he is in possession of the private part of the *Production Key*.

In the *produced* state the boot loader accepts the following commands:

Command	Description	Authentication necessary?
GetState	ask for information about state, hardware and temperature	no
BLSetRTC	set the Real Time Clock of the CryptoServer	yes
BLGetAlarmLog	ask for the alarm history	no
BLGetTempLog	ask for the temperature history	no
BLResetAlarm	reset of a previous alarm (see above)	yes
BLLoadInitKey	load the public part of the *Module Signature Key* $\mathbf{K_{MDLSIG\text{-}PUB}}$ as well as of the *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$	yes
BLClear	clear the CryptoServer to the state • *produced* • *manufactured*	yes
BLChangeEID	Change ADM1 field of the EID.	yes
Update	Update boot loader code.	yes

For the performance of the sensitive commands *BLSetRTC*, *BLResetAlarm*, *BLLoadInitKey*, *BLClear, Update* and *BL ChangeEID* authentication via signature done by the *Production Key* $\mathbf{K_{PROD\text{-}PRV}}$ is necessary.

CryptoServer's produced *state exclusively occurs at the manufacturer's site and never at the customer's site. Therefore in particular the* Initialization Key $\boldsymbol{K_{INIT\text{-}PUB}}$ *cannot be loaded by the customer himself.*

3.2.4 State: Initialized

As soon as the public part of the *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$ has been loaded with the boot loader command *BLLoadInitKey* (which can only be performed in the *produced* state and only by the manufacturer who has previously authenticated himself with the *Production Key*), the CryptoServer's state is set to *initialized*. If the *Initialization Key* is customer-specific, at this point for the first time a direct connection is set up between CryptoServer and client.

The CryptoServer state initialized *also occurs at the customer's site. Command authentication is now additionally possible with the help of a RSA signature of the command which is done by the private part of the* Initialization Key $\boldsymbol{K_{INIT\text{-}PRV}}$, *and which can be verified by the boot loader using the public part* $\boldsymbol{K_{INIT\text{-}PUB}}$.

The CryptoServer is already secured in this manner on its way from the manufacturer to the customer. In particular, "foreign" software cannot be loaded onto the CryptoServer unnoticeably.

Using the *BLLoadInitKey* command the manufacturer's *Module Signature Key* $\mathbf{K_{MDL\ SIG\ PUB}}$ (its public part) has been additionally loaded. Once the *Initialization Key* has been successfully loaded, the CryptoServer-specific local Master Key $\mathbf{K_{CS2}}$ will internally be generated and saved in the sensory-protected Key-RAM. For additional information on the importance and use of the various keys please go to chapter 3.6.

The following (boot loader) commands will be accepted by the CryptoServer if the boot loader is active (i. e. the CryptoServer is in the so-called *boot loader mode*, see 2.3.4) and if the CryptoServer is at least set on the *initialized* state:

Command	Description	Authentication necessary?
GetState	Require for information about status, hardware and temperature	no
BLResetAlarm	Resetting a previous alarm (see 3.3) (executable only if previously an alarm – which is not already reset – has occurred)	yes (with $\mathbf{K_{INIT\text{-}PRV}}$ or $\mathbf{K_{PROD\text{-}PRV}}$)
BLClear	Deleting the CryptoServer to the state ■ *produced* (command must be signed with $\mathbf{K_{PROD\text{-}PRV}}$) ■ *manufactured* (command must be signed with $\mathbf{K_{PROD\text{-}PRV}}$) ■ *initialized* (command must be signed with $\mathbf{K_{INIT\text{-}PRV}}$) (see also 3.5)	yes (see left)

Command	Description	Authentication necessary?
BLSetRTC	Setting the CryptoServer's internal clock (Real Time Clock)	yes ($\mathbf{K}_{\mathbf{INIT\text{-}PRV}}$)
GetTime	Getting CryptoServer's internal time	no
BLChangeInitKey	Changing the public part of the Initialization Key $\mathbf{K}_{\mathbf{INIT\text{-}PUB}}$	yes ($\mathbf{K}_{\mathbf{INIT\text{-}PRV}}$)
BLLoadFile	File loading onto the CryptoServer	yes ($\mathbf{K}_{\mathbf{INIT\text{-}PRV}}$)
ListFiles	Request for a list of all loaded files	no
GetBootLog	Request for info about boot procedure	no
GetAlarmLog	Request for alarm history	no
GetTempLog	Request for temperature history	no
GetTimeLog	Request for history about time changes/settings	no
StartOS	Starting the SMOS operating system (from the FLASH directory)	no
RecoverOS	Starting the back-up copy of the SMOS operating system (from the SYS directory)	no
Test	Execute communication test with CryptoServer	no

Thereby security-relevant commands (i. e. all commands which have to be authenticated) have to be signed with the *Initialization Key* $\mathbf{K}_{\mathbf{INIT\text{-}PRV}}$. *BLResetAlarm* can optionally be signed by the manufacturer with the *Production Key* $\mathbf{K}_{\mathbf{PROD\text{-}PRV}}$.

The *BLClear* command must be signed with the *Production Key* if the CryptoServer is to be deleted to the *produced* or *manufactured* state.

Even if the operating system and further firmware modules are already loaded, the CryptoServer can still be in the *initialized* state: As long as during the boot process of the CrytoServer the *Initialization Key* is found but the operating system SMOS has not yet been started (i. e. the boot loader is still active), the CryptoServer remains in the *initialized* state.

If the CryptoServer is in boot loader mode *(i. e. the boot loader is still active) it can at most observe the global state* initialized*, independently from the actually loaded firmware.*

3.2.5 State: Operational

When receiving an *initialized* CryptoServer, the customer is able to load the operating system module SMOS and the basic firmware modules (see 2.3.2) containing the base functionality for download and communication. This *BLLoadFile* boot loader command has to be signed with the private part $\mathbf{K_{INIT\text{-}PRV}}$ of the customer's *Initialization Key*.[4] At the end of that, the OS can be started (boot loader command *StartOS* which does not have to be authenticated). If this has been successfully completed, the boot loader terminates and the CryptoServer reaches the *operational* state.

From now on, each time the CryptoServer reboots, if the boot loader finds the CryptoServer at least in the *initialized* state (i. e. it finds the public *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$) and if the boot loader is able to start the OS successfully at the end of the boot procedure, the global state of the CryptoServer is considered to be *operational*.

This *operational* state does not say anything about the available external functionality: to have the full spectrum of the external interface of the CryptoServer, the appropriate firmware modules have to be additionally loaded. Once the SMOS operating system has been successfully initialized, it will automatically search for further available firmware modules in the flash directory and subsequently start them.

If in the CryptoServer apart from the SMOS operating system there are also the further basis firmware modules loaded (ADM, CMDS and UTIL), and if it is possible to successfully start all these modules, the CryptoServer will be in the so-called ***operational mode****, see 2.3.4. This means that it is ready for its "operational work". Sensitive commands must still be authenticated. But in operational mode much more possibilities exist for this purpose than in boot loader mode, see chapter 2.6. Nevertheless, administration commands can be further on authenticated with the* Initialization Key.

In the *operational* state the boot loader is no longer activ, instead, commands are then processed by the CMDS module (Command Scheduler) – if available. Executable commands are forwarded by CMDS to the corresponding external interfaces (functions) of the firmware modules loaded into the CryptoServer. It depends on the loaded firmware modules which commands are executable now (see 4.8-4.11 for the basic administration commands).

Contrary to all the other states, the operational *state represents a status which is unidentifiable by the boot loader as the latter is not active anymore at this point of time. If the SMOS operating system is loaded but not started, CryptoServer is further considered to be in the* initialized *state.*

[4] At this point the customer has an automatic control if really his public key $\mathbf{K_{INIT\text{-}PUB}}$ was loaded by the manufacturer into the CryptoServer: if this was not the case, the *BLLoadFile* command, signed with his $\mathbf{K_{INIT\text{-}PRV}}$, would not work.

3.2.6 State: Defect

If the boot loader's self test fails during the starting phase, the CryptoServer will be in the *defect* state. This test is always run at the beginning of the boot phase after the deletion of the IRAM.

In the defect state the CryptoServer exclusively accepts the commands *GetState* and *GetBootLog* (if yet technically possible), which are not to be authenticated. The alarm mechanism of the CryptoServer remains unchanged.

If the CryptoServer is in the defect *state, please contact the manufacturer/Utimaco.*

3.3 Alarm

The *Alarm* state of the CryptoServer has not to be mixed up with the above mentioned global states (blank, manufactured, produced, initialized, operational, defect) the CryptoServer can be in. Anytime a physical alarm happens to the CryptoServer, it will immediately be detected by the sensory which triggers the defined alarm mechanism (see below for details). Part of this mechanism is a restart of the CryptoServer. *Alarm state* is given if the boot loader detects an alarm condition in the alarm status register during the boot process. The CryptoServer responds with the current alarm condition and a *BLResetAlarm* command is required to reset the alarm status register (see 7.2 and 0).

Generally, two different kinds of alarm are possible:

- temporary alarms
- permanent alarms

where only temporary alarms can be reset. Permanent alarms cannot be reset. Permanent alarms occur in case of a damage of the inner or outer tamper protecting foil, whereas usually all other possible alarms are temporary.

Possible reasons for alarm are

Abbreviation	Explanation
Temp_low	Temperature too low (see also 3.4)
Temp_high	Temperature too high (see also 3.4)
In_foil	Internal foil damaged
Out_foil	External foil damaged
Pow_high	Voltage / tension too high
Pow_low	Voltage / tension too low
ext_Erase	External deleting / clearing done
inval_MK	Invalid (corrupted) Master Key $\mathbf{K_{CS2}}$ (reason for this is usually an empty battery, see below)

Whenever a physical alarm occurs (noticed by the sensory which watches temperature, voltage and chemical or mechanical attack, i. e. destruction of the foil), the *Alarm-Bit* in the alarm status sensory register will be set immediately, together with bits which indicate the kind of alarm.

To demonstrate that the alarm still has to be logged a second bit, the *Register-Valid-Bit*, will be set by the sensory, too. The CryptoServer's internal master key $\mathbf{K_{CS2}}$ will be deleted (i. e. the Key-RAM will be erased by hardware) and the CryptoServer will be restarted. Independently from the occurrence of an alarm, the DSP IRAM will be erased at the very beginning of the boot procedure. Clearing of Key-RAM and IRAM will be finished within less than 4msec after the occurrence of the alarm.

If during the (following) boot process the boot loader finds an alarm in the alarm status register which is not yet logged (i. e. the *Register-Valid-Bit* is still set), at first data and firmware will be erased in order to set the CryptoServer back to the *initialized* state:

In case of an alarm, all firmware modules inside the CryptoServer will be deleted, as well as all customer's data except the public Initialization Key*.*
The CryptoServer is now in the initialized *state.*

After an alarm, the CryptoServer contains only the boot loader code and no further firmware. Furthermore, the manufacturer's public *Production Key* $\mathbf{K_{PROD-PUB}}$ and the customer's public *Initialization Key* $\mathbf{K_{INIT-PUB}}$ (together with the manufacturer's public key for firmware signature $\mathbf{K_{MDLSIG-PUB}}$) as well as the 'alarm.log' file and the configuration file 'bl.ini' will remain but no other customer-related keys and data. In particular, CryptoServer's local master key $\mathbf{K_{CS2}}$ is erased.[5]

Afterwards the boot loader adds up an entry provided with timestamp into the log file 'alarm.log', in which also the alarm cause is stated. This file can anytime and in any mode (boot loader mode or operational mode) be read out with the help of the GetAlarmLog *command.*

If in this moment the physical alarm is not present any more (i. e. the cause of the alarm was corrected in the meanwhile, e. g. an old battery has been already replaced by a fresh one in case of a low-power-alarm) the boot loader continues with the boot process.

Each alarm must explicitly be reset by the user using the BLResetAlarm *command before the boot loader accepts any further commands. By the fact that this* BLResetAlarm *command must be authenticated with the* Production *or* Initialization Key*, it is ensured that in case of each alarm occurred at least one responsible person has been informed about.*

With *BLResetAlarm* the Alarm-Bit and the Register-Valid Bit will be set back (to demonstrate that the alarm has already been logged) and the boot process will continue.

If the alarm cause has not been remedied before the BLResetAlarm *resetting, hardware will trigger a renewed CryptoServer's reset and the procedure will be repeated.*
Therefore a permanent alarm, e. g. foil damages, cannot be reset, in which case you are required to contact the manufacturer/Utimaco.

If the CryptoServer is stored for a long period of time without voltage supply and the battery gets empty, the CryptoServer-own master key $\mathbf{K_{CS2}}$ will then be deleted (correctly according to the alarm mechanism); however, information about this alarm state will be

[5] A new $\mathbf{K_{CS2}}$ will be generated when the next *BLResetAlarm* command will be performed.

lost, too, as the content of the sensors register is also lost in the absence of voltage. Therefore the boot loader additionally checks at each boot process the integrity of the loaded master key $\mathbf{K_{CS2}}$. If the verification fails, the boot loader will then activate a 'virtual' alarm. This alarm is displayed as 'Inval_MK', see above.

For the accurate procedure "What to do?" in case of an alarm see chapter 7.2.

3.4 Behavior of CryptoServer outside the Normal Temperature Range

If the internal temperature of the CryptoServer gets outside of the normal operating temperature range, the CryptoServer will behave in a special way, as shown in the table below:

Temperature	Behavior of the CryptoServer
below –13°C	An alarm is triggered and the CryptoServer enters *power down mode*.
–13°C to 5°C	The CryptoServer enters power down mode.
5°C to 58°C	Normal operation.
58°C to 66°C	The CryptoServer enters power down mode.
above 66°C	An alarm is triggered and the CryptoServer enters power down mode.

All temperature values in the table are approximate values. The exact temperature values may vary a little because of tolerances of the electronic components and the use of a hysteresis by the comparators. Details can be seen from the “CS2000 Power / Temperature Sensory Specification”.

Note that only the temperature inside the inner case of the CryptoServer device is relevant, not the environment temperature. The actual value of the inner temperature can be retrieved with the GetState *administration command.*

Once the CryptoServer has entered power down mode, it does not respond to any request. An attempt to access the CryptoServer will usually result in a kind of timeout error from the device driver.

Before entering power down mode the boot loader writes an entry into the temperature log file which can be read out with the administration command *GetTempLog* (see 4.7.13 and 4.8.10).

The only way to get the CryptoServer out of the power down mode is to reset it using one of the *Restart, Reset* or *ResetToBL* commands.

Resetting the CryptoServer has no effect, if the temperature is still outside the normal range. In case of CryptoServer’s power down caused by high temperature, it is recommended to switch the supply power off for some time in order to cool the CryptoServer down.

3.5 Clear Commands

The boot loader clear commands (*BLClear*, see 4.7.4) are required to give you the possibility to erase the CryptoServer without initiating an alarm condition. They perform all actions necessary to get the CryptoServer back into a previous state.

The customer can perform the BLClear *command only with the option to clear the CryptoServer to the* initialized *state. This command has to be signed with the customer's* Initialization Key $K_{INIT\text{-}PRV}$. *It can only be performed in boot loader mode.*

The *Clear-to-Initialized* command will erase

- CryptoServer's local master key K_{CS2} (by generating and storing a new K_{CS2}) and will
- (implicitly) erase the operating system SMOS and all firmware modules (only the boot loader code remains) as well as
- all data except for the public keys $K_{PROD\text{-}PUB}$ and the $K_{INIT\text{-}PUB}$ (together with the $K_{MDL\text{-}SIG\text{-}PUB}$), the alarm log file and the boot loader configuration file 'bl.ini'.

After that a reboot will be performed and the CryptoServer will stop in the *initialized* state.

There are further *Clear* commands with other erase options: *Clear-to-Produced* respectively *Clear-to-Manufactured*, where more data like e. g. the customer's *Initialization Key* $K_{INIT\text{-}PUB}$ and the manufacturer's *Module Signature Key* $K_{MDLSIG\text{-}PUB}$ are erased so that the CryptoServer falls back to the *produced* and *manufactured* state respectively. These commands must be signed by the manufacturer's private *Production Key* $K_{PROD\text{-}PRV}$ and can thus be performed only by the manufacturer.

3.6 System Keys

This chapter will describe different keys used in and around a CryptoServer, how they are generated, who will be responsible for storing them, and the way they are handled and used.

3.6.1 Manufacturer's Production Key K_{PROD}

type:	The Production Key is a RSA key and at least 1024 bit long.
naming:	private Production Key: **$K_{PROD-PRV}$** public Production Key: **$K_{PROD-PUB}$**
generation:	The generation of the key pair will be done by the manufacturer himself.
responsibility:	The key is manufacturer-specific (but usually not CryptoServer-individual). The manufacturer is responsible for using and storing the key.
life cycle:	The private part of the production key **$K_{PROD\ PRV}$** will be stored in a safe environment at the manufacturer's site where only authorized personnel has access. The public part of the key **$K_{PROD\ PUB}$** will be imported into the CryptoServer by the manufacturer (using the *BLLoadProductionKey* command in the *manufactured* state). It will be stored in the CryptoServer's flash file. Only the boot loader has write access to the key. From this point on (state at least *produced*) the key **$K_{PROD-PUB}$** is resident inside the CryptoServer until the *Clear-to-Manufactured* command is performed. This can only be performed by the manufacturer himself.
usage:	The Production Key is needed to prevent unauthorized access immediately after the CryptoServer's production. It has to be used to assign a CryptoServer to a customer (initialization): the *BLLoadInitKey* command has to be signed with the **$K_{PROD\ PRV}$**. The key can be used to clear the CryptoServer to the *produced* or *manufactured* state (*Clear-to-Produced* or *Clear-to-Manufactured* command), to set the RTC (if the CryptoServer is in the *produced* state), to change the EID, to update the boot loader code and to perform the *BLResetAlarm* command. **No further usage is defined**

The Production Key will not be used by the customer.

3.6.2 Customer's Initialization Key K_{INIT}

type:	The Initialization Key is a RSA key and at least 1024 bit long.
naming:	private Initialization Key: **$K_{INIT-PRV}$** public Initialization Key: **$K_{INIT-PUB}$**
generation:	Generation can be done by the customer himself (e. g. using Utimaco's Key Tool). The customer will hand over the public key to the manufacturer.
responsibility:	The key is customer specific (but usually not CryptoServer-individual). The customer is responsible for using and storing both key parts. The manufacturer is responsible for the import of the public part of the key into the CryptoServer after the production process.
life cycle:	The public part **$K_{INIT\ PUB}$** of the key will be imported into the CryptoServer by the manufacturer (using the boot loader command *BLLoadInitKey* in the *produced* state, see 4.7.6). Once a CryptoServer is assigned to a customer the key **$K_{INIT-PUB}$** is resident inside the CryptoServer's flash file. Only with specific *BLClear* commands (*Clear-to-Produced*, *Clear-to-Manufactured*) the key may be cleared inside the CryptoServer. In this case the CryptoServer has to be re-initialized; this must be done by the manufacturer. **$K_{INIT-PUB}$** can be changed by the customer via the *BLChangeInitKey* boot loader command, see 4.7.5. This command must be signed with the old Initialization Key **$K_{INIT-PRV}$**. This change makes only sense if it is performed together with the *BLClear* command (because otherwise the still loaded firmware modules could no longer be started).
usage:	The Initialization Key is needed to prevent unauthorized access after CryptoServer's initialization (e. g. during shipment). **In boot loader mode**, the key must be used to authenticate the security relevant commands sent to an *initialized* CryptoServer: *BLLoadFile, BLSetRTC, BLResetAlarm, BLChangeInitKey* and *BLClear* (*Clear-to-Initialized*) will only be performed by the CryptoServer if they are signed by the **$K_{INIT-PRV}$** (*BLResetAlarm* can optionally be signed with the **$K_{PROD\ PRV}$** instead). **In operational mode**, the Initialization Key can be used to authenticate the security relevant administrative commands: The commands for loading, replacement and deletion of firmware modules, *SetRTC*, creating or deleting a user or changing the user rights in the user database will be performed by the CryptoServer if they are signed with the **$K_{INIT-PRV}$** by the predefined **user *ADMIN***, see 2.6.

	Attention: On this basis it is possible to define further users with other authentication mechanisms (*CreateUser* command) who would by then also get the permission to perform these administrative commands. Furthermore, the $\mathbf{K_{INIT\text{-}PRV}}$ key is used during the preparation of a firmware module: With this key the customer-based MTC signature before firmware download has to be done, see 3.7. The public key $\mathbf{K_{INIT\text{-}PUB}}$ inside the CryptoServer will be used to verify the MTC signature of a loaded module every time the module is started (e. g. after a reset). Otherwise the firmware module will not be started. Further customer-defined usage of the Initialization Key is possible.
recommendations:	The private part of the Initialization Key $\mathbf{K_{INIT\ PRV}}$ must be stored in a safe environment at the customer's site where only authorized personnel has access. The Initialization Key should be changed for each new application or project.

If the customer has lost his private Initialization Key $\boldsymbol{K_{INIT\text{-}PRV}}$, he is not able to administrate the CryptoServer any more. The CryptoServer has to be sent back to the manufacturer. It will be cleared and a new Initialization Key will be loaded using the manufacturer's Production Key.

3.6.3 Manufacturer's Module Signature Key $K_{MDL-SIG}$

type:	The Module Signature Key is a RSA key and at least 1024 bit long.
naming:	private Module Signature Key: **$K_{MDL-SIG-PRV}$** public Module Signature Key: **$K_{MDL-SIG-PUB}$**
generation:	Generation will be done by the manufacturer himself.
responsibility:	The manufacturer is responsible for using and storing the key pair.
life cycle:	The private part of the Module Signature Key **$K_{MDL-SIG-PRV}$** will be stored in a safe environment at the manufacturer's site where only authorized personnel has access. The public part of the key **$K_{MDL-SIG-PUB}$** will be imported into the CryptoServer by the manufacturer during initialization (using the *BLLoadInitKey* command in the *produced* state). From this point on (state at least *initialized*) the key **$K_{MDL-SIG-PUB}$** is resident inside the CryptoServer's flash file. Only with a specific *BLClear* command (*Clear-to-Produced, Clear-to-Manufactured*) the key might be cleared inside the CryptoServer (together with the Initialization Key). In this case the CryptoServer has to be re-initialized by the manufacturer. The **$K_{MDL-SIG-PUB}$** cannot be changed by the customer.
usage:	The private key **$K_{MDL-SIG-PRV}$** is used by the manufacturer to sign its MMC's (module manufacturer containers), see 3.7. A MMC contains the executable of a CryptoServer firmware module. The MMC signature is used to check the authenticity of the MMC at a later time (i. e. to verify the origin by the manufacturer). The public key **$K_{MDL-SIG-PUB}$** is resident inside the CryptoServer. Every time before starting a firmware module (e. g. after a reset), the operating system SMOS will automatically verify the manufacturer's signature of the MMC with the **$K_{MDL-SIG-PUB}$**. Otherwise the firmware module will not be started.

The Module Signature Key will not be used by the customer.

3.6.4 CryptoServer's Local Master Key K_{CS2}

type:	On default this key is a 32 bytes AES key. If no AES functionality is loaded into the CryptoServer, a 24 bytes triple DES key is used instead.
naming:	CryptoServer's local master key **K_{CS2}**
generation:	Generation will be done automatically inside of and by the CryptoServer itself after successful initialization, i. e. after downloading the customer's public Initialization Key. **K_{CS2}** is CryptoServer-individual and will never leave the CryptoServer.
responsibility:	The CryptoServer security module itself is responsible for key usage, storage and deletion.
life cycle:	After having been generated, the key is stored inside of CryptoServer in a small, sensory protected RAM (see *Key-RAM* 2.1.7.6). The life cycle of this key ends up when an alarm occurs to the CryptoServer. If any alarm cause (temperature, foil, voltage) is detected by CryptoServer's sensory, the memory area in which the key **K_{CS2}** is stored will be automatically erased. A new **K_{CS2}** is generated during the next normal boot process (after successful performance of the *BLResetAlarm* command). The key is also erased if a *Clear-to-Produced* or *Clear-to-Manufactured* command is sent to the boot loader. This resets the CryptoServer into the *produced* or *manufactured* state, while a new **K_{CS2}** is generated during CryptoServer's next initialization. If the *Clear-to-Initialized* command is performed, a new **K_{CS2}** will be immediately generated and overwrite the old one.
usage:	The local master key **K_{CS2}** is needed to encrypt data inside the CryptoServer.[6] For this purpose it will be used internally by other firmware modules. There is a database firmware module (module DB) which facilitates the usage of the key. This DB module provides an internal public interface for secure data storage to other application firmware modules. DB will use the **K_{CS2}** in full length. **K_{CS2}** will never be available outside the CryptoServer.

*The customer will never use CryptoServer's local master key **K_{CS2}** directly. The CryptoServer itself is completely responsible for generation, usage and erasure (in case of alarm) of the **K_{CS2}***

[6] This is necessary in order to be able to store sensitive data also in memory areas which are not sensory-protected, i. e. which will not be erased within a very short time after an alarm has occurred. These are e. g. the flash device, NV-RAM and SD-RAM.

3.6.5 Firmware Delivery Key K_{DELV}

type:	The Firmware Delivery Key is a RSA key and at least 1024 bit long.
naming:	private Firmware Delivery Key: **$K_{DELV-PRV}$** public Firmware Delivery Key: **$K_{DELV-PUB}$**
generation:	Generation will be done by the firmware developer himself.
responsibility:	The developer of a firmware module is responsible for using and storing the key pair. He has to give the public key to the customer who should in turn protect it against the possibility of unauthorized exchange.
usage:	The private key **$K_{DELV-PRV}$** is used for the signature of delivery MTC's (module transport container), see 3.7. A MTC contains the MMC (module manufacturer container) with a CryptoServer firmware module. The MTC signature is used to check the integrity and authenticity of the delivered firmware module. Before such a MTC can be downloaded into the CryptoServer the customer has to verify the signature and replace it by his own signature calculated with the customer's Initialization Key **$K_{INIT-PRV}$** (for this procedure, see 6.2, 6.3 and 3.7.1.2).
life cycle:	The key is not directly linked to the CryptoServer's life cycle. It can only be used to guarantee the authentic transport of firmware modules from the developer to the customer. No part of the key is stored inside the CryptoServer.

The customer has to protect the public Firmware Delivery Key against the possibility of unauthorized exchange (because otherwise he could be deceived about the origin of a MTC)

3.7 Firmware Module Management

In this section the management of the CryptoServer's firmware modules is described from a security point of view.

Before CryptoServer firmware can run inside the CryptoServer it is transported twice: first from the firmware developer[7] to the customer who wants to use the firmware inside his CryptoServer, second from the customer into his CryptoServer. With the aim to link the firmware with administrative information (e. g. compilation date, version number etc.) and to add signatures to check the authenticity and integrity of the firmware the firmware is enveloped into so called *containers*.

For easier handling, Utimaco offers also so-called *package files* in which a set of firmware modules (the containers described below) is bundled, ready for download. See 3.7.2 for the concept and usage of these firmware packages.

3.7.1 Firmware Containers: MTC, MMC

The *firmware module* itself is the executable module compiled for the CryptoServer platform, i. e. ready for interpretation of the CryptoServer's operating system SMOS (COFF file '*.out').

Before such a module can be downloaded into the CryptoServer it must be enveloped twice:

- first the module is enveloped into a **MMC** (*module manufacturer container*),
- then this MMC is enveloped into a **MTC** (*module transport container*).

The **MMC** adds a header with general information about the firmware module (like manufacturer name, module name, module ID, compilation date, version number etc.) and a signature that guarantees the authenticity of the module (origin by manufacturer/-developer). The MMC signature, done with the manufacturer's private key for module signature $\mathbf{K_{MDL\text{-}SIG\text{-}PRV}}$ (whose public opponent is stored within the CryptoServer, see 3.6.3), is mandatory: the MMC signature of a firmware module will be checked every time the CryptoServer is (re)started, if it cannot be verified, the firmware module will not be started.

The **MTC** adds a second header (which contains additional module transport information) and a second signature which is calculated over the complete MMC. The MTC signature can be used for two different purposes:

- first (optionally) to secure the transport of the module from the developer to the customer (to guarantee the module authenticity and integrity during delivery),
- second (mandatory) to give the CryptoServer the possibility to check whether a downloaded module is authentic from the customer the CryptoServer belongs to.

[7] Such firmware modules may be written either by the manufacturer or the customer or a third party developer. In this section the firmware module creator is called *developer* regardless wether he belongs to the manufacturer, customer or a third party infrastructure.

These transport mechanisms will be described below.

The following picture shows the model of a downloadable MTC:

<table>
<tr><td></td><td colspan="3">Module Manufacturer Container – MMC
(*.mmc, generated by manufacturer/developer)</td><td></td></tr>
<tr><td>MTC Header</td><td>MMC Header</td><td>Binary
(*.out, from developer)</td><td>MMC Signature
(done with Module Signature Key)</td><td>MTC-Signature
(done with Initialization Key)</td></tr>
<tr><td colspan="5">Module Transport Container – MTC
(*.mtc, generated by customer)</td></tr>
</table>

3.7.1.1 Firmware Delivery from Developer to Customer

The MTC structure can optionally be used to secure the delivery of software modules from the developer (who does not have to be identical with the manufacturer) to the customer.

For this, the firmware developer has to use his firmware delivery key $\mathbf{K_{DELV}}$ (see 3.6.5) where the public part $\mathbf{K_{DELV\text{-}PUB}}$ is given to the customer. This key guarantees the authenticity of the module during transport.

To generate the MTC, the developer envelopes the firmware MMC into a MTC by creating a MTC header with administrative information and by adding the MTC signature. This is calculated with the private part of his firmware delivery key $\mathbf{K_{DELV\text{-}PRV}}$. This signature is done over the complete MMC, using the *MakeMTC* command of the CSADM tool, see 4.5.1.

The MTC is then shipped to the customer. The MTC developer signature can be verified by the customer with the public part of the developer's Firmware Delivery Key $\mathbf{K_{DELV\text{-}PUB}}$, using the *VerifyMTC* command of the CSADM tool, see 4.5.3. Only if this signature can be verified successfully the MTC should be accepted by the customer and can be added to the reliable pool of the customer's firmware modules.

3.7.1.2 Download of Firmware Modules onto the CryptoServer

For being downloaded onto the CryptoServer a firmware module has to be enveloped into a MTC. MMC signature and MTC signature will be checked during the downloading procedure itself

If a firmware module shall be downloaded onto a CryptoServer, the MTC signature (as well as the MMC signature) is mandatory. The MTC signature has to be done by the customer himself with the private part of his Initialization Key $\mathbf{K_{INIT\text{-}PRV}}$ (see 3.6.2). With the public counterpart $\mathbf{K_{INIT\text{-}PUB}}$ of the key, the CryptoServer as recipient of the firmware module can check during firmware download whether the module comes authentic from the customer the CryptoServer belongs to. Additionally, this signature verification will be done every time the CryptoServer starts the module (after a reset/restart); and with this verification the module integrity is automatically proven, too.

If the MTC structure has been used before to secure the transport of the firmware module from the developer to the customer, it is most important to verify the developer's signature on the MTC first (with the firmware delivery key $\mathbf{K_{DELV\text{-}PUB}}$ and the *VerifyMTC* command, see subsection above). Afterwards this developer's signature has to be removed (using the *RemoveMTC* command of the CSADM tool, see 4.5.2).

To prepare the firmware module for the download onto his CryptoServer, the customer now has to re-sign the MTC with his Initialization Key $\mathbf{K_{INIT\text{-}PRV}}$ (*MakeMTC* command, see 4.5.1). The enveloped MMC is not (and should not be!) altered during this procedure.

If CryptoServer's Initialization Key is changed, all loaded firmware modules will have to be reloaded, with new MTC signatures.

In this case, the old MTC signatures (done with the old Initialization Key) have to be removed again (*RemoveMTC* command) and the new MTC signatures have to be generated with the private part of the new Initialization Key (*MakeMTC* command).

For simplified firmware management *package files* and the related commands for package download, verification and re-signing can be used, see chapter below.

3.7.2 Package Files

In order to simplify firmware management Utimaco offers firmware also in so-called *package files*: A package file "*.mpkg" can contain several firmware modules (e. g. MTCs) as well as other files in a packed form.

Package files are intended to give the user a facile possibility to download or update a set of several firmware modules and/or files into the CryptoServer in only one step.

For this purpose the CryptoServer's administration tool CSADM offers the *LoadPkg* command (see 4.8.15) which can replace a series of succeeding CSADM commands:

The *LoadPkg* command (which has to be signed by the CryptoServer's Initialization Key) loads the contents of the given package file "*.mpkg" into the CryptoServer, adding to or

replacing existing firmware. In case that firmware modules contained in the package file are not signed with the given Initialization Key, these MTC files will automatically be (re-)-signed before being loaded into the CryptoServer. To see in detail how CryptoServer's (re-)initialization or firmware update can be simplified that way, see also 6.2 and 6.4.

Furthermore the CSADM tool offers commands to create a package file or to list its contents, to retrieve the contained modules from a "*.mpkg" file, and to compare the contents of a package file with the firmware stored in the CryptoServer (commands *Pack* 4.5.6, *ListPkg* 4.5.8, *Unpack* 4.5.7, *CheckPkg* 4.8.16). Additionally you can also check the MTC and MMC signature of each firmware module contained in a package (*VerifyPkg* 4.5.9) and re-sign each module with a new Initialization Key (*ResignPkg* 4.5.10).

3.8 Delivery of CryptoServer Systems

Customer's usage of firmware containers depends on the delivered CryptoServer's type, i. e. whether this is a test system or a production system.

3.8.1 CryptoServer Test System

To speed up development and test on the customer's side, Utimaco offers ready-to-use CryptoServer systems already prepared for using a publically known Initialization Key.

A CryptoServer test system is already loaded with firmware modules at the time of Utimaco's delivery to the customer. CryptoServer's loaded Initialization Key comes from the manufacturer Utimaco. The customer receives a copy of its private part in order to be able to perform important administration functions.

If further firmware is to be loaded onto the CryptoServer (software updates or additional modules), the module will then be developed by the manufacturer and packed in the corresponding MMC and MTC containers; the *downloadable* MTC will afterwards be delivered to the customer, being signed with the manufacturer's Initialization Key. The customer can directly load the MTC into the CryptoServer using a corresponding administration command (authenticated e. g. by the 'ADMIN' user with a signature from the Initialization Key), see 6.4.

3.8.2 CryptoServer Production System

A CryptoServer production system uses the customer's individual Initialization Key. If loaded or replaced, the firmware must be packed in a MTC, which has been previously signed by the customer with his Initialization Key.

The firmware module delivered by the manufacturer/developer can have one of the following formats:

1. raw software module (file '*.out')
2. MMC (signed with the module signature key $\mathbf{K_{MDL\text{-}SIG\text{-}PRV}}$)
3. MTC (signed with a developer's Firmware Delivery Key).

In case 1 and 2, the customer must envelope the '*.out' and '*.mmc' files respectively into a MTC using the corresponding *MakeMTC* command of the Administration Tool CSADM, see 4.5.1.

In case 3 the MTC has to be re-signed, i. e. the developer's MTC signature is to be successfully verified, removed and replaced by a new one made with customer's private Initialization Key. For this purpose the corresponding commands of the Administration Tool have to be performed, see 4.5.

Please keep in mind that in case that a firmware package file "*.mpkg" containing MMCs and MTCs is used instead of a set of single firmware modules, these steps will be

performed automatically during firmware download with the *LoadPkg* command (see 3.7.2). Only the verification of the firmware's authenticity (e. g. verification of developer's MTC signature on each firmware module of the package file) has to be done manually by the customer: for this please use the *VerifyPkg* command, see 4.5.9.

3.8.3 How to Change a CryptoServer Test System in a Production System

The customer can convert a CryptoServer hitherto used as a test system (with the Initialization Key of Utimaco) into a production system by resetting the CryptoServer to the *initialized* state (i. e. running the *BLClear* command, 4.7.4) and subsequently changing the Initialization Key (*BLChangeInitKey* command, see 4.7.5), i. e. replacing Utimaco's key by a personal one.

As with *BLClear* all firmware modules and application-specific data are deleted, all necessary firmware is afterwards to be reloaded. The MTC signatures must previously be re-calculated for all firmware modules that shall be reloaded, using the new customer-own Initialization Key (i. e. remove the old MTC signature with *RemoveMTC* and then re-sign the modules with the new Initialization Key, using the *MakeMTC* command).

In case the firmware modules are given in form of a package file "*.mpkg", it is not necessary to do this manually for each MTC: The *LoadPkg* command for firmware download will include the re-signing of all MTCs of the package file if the flag `ForceReSign` is given (see 4.8.15). To check the firmware's authenticity before (e. g. verification of developer's MTC signature on each firmware module of the package file) the *VerifyPkg* command can be used, see 4.5.9.

For details, see sections 6.4 and 6.5.

4 The CryptoServer Administration Tool CSADM

The *CryptoServer Administration Tool* (CSADM) is a command line utility designed for being called from the command line or in a batch file. It offers functions either to execute commands on the CryptoServer (addressing the boot loader, administration module or CMDS module) or the CryptoServer LAN. In addition it contains utility functions which will be processed without a connection to a CryptoServer (e. g. preparing of firmware modules).

The following requirements have to be made for the CSADM:

PC-Hardware:

- no special requirements as far as memory and CPU performance is concerned.
- network interface card to access CryptoServer LAN
- one free serial port to connect the PIN-Pad (smart card reader with keyboard and display).

PC-Software (OS):

- Windows NT or
- Winows XP or
- Windows 2000/2003 or
- Linux.

4.1.1 Installation of the CSADM

The installation of the Administration Tool CSADM is quite simple.

Installation under Windows:

1. Copy the file ‘csadm.exe’ to a well-chosen directory.
2. Add this directory to the ‘PATH’ environment variable to be able to call the Administration Tool from any other directory.
3. If you do not want to set the ‘Dev=’ parameter with each execution of a CSADM command: It is possible to set an environment variable CRYPTOSERVER (e. g. with the value ‚PCI:0‘) which sets the CryptoServer address permanently (unless a ‘Dev=’ parameter is explicitly set for a specific command, see 4.1.2).

Installation under Linux:

1. Copy the executable file ‘csadm’ to a well-chosen bin directory.
2. If you do not want to set the ‘Dev=’ parameter with each execution of a CSADM command: It is possible to set an environment variable CRYPTOSERVER (e. g. with the value ‚/dev/cs2‘) which sets the CryptoServer address permanently (unless a ‘Dev=’ parameter is explicitly set for a specific command, see 4.1.2).

4.1.2 Syntax of the CSADM

The syntax of a CSADM command is according to the following scheme:

csadm [Dev=...] <param1>[=...] <param2>[=...] ... <command1>[=...] <command2>[=...] ...

Note:

- *Parameters and commands are processed from left to right.*
- *If a subsequent command requires to set a parameter or to run another command prior to its own execution, it will have to be entered rightmost.*
- *Expressions in squared brackets [ ] are optional.*
- *In the syntax describing line of the individual command descriptions all parameters are shown in angle brackets < > and are explained later.*
- *Some parameters or commands require an assigned value ('=...'), some do not.*
- *Some commands use a default value if none is given ('[=...]').*

Examples:

- csadm MTCPubKey=c:\keys\init_dev_pub.key VerifyMTC=exmp.mtc
- csadm MTCSignKey=:cs2:cp8:COM1 RemoveMTC=exmp.mtc MakeMTC=exmp.mmc
- csadm Dev=PCI:0 GetState
- csadm Dev=TCP:192.168.1.98 InitPrvKey=d:\keys\cs2\init_dev_prv.key BLLoadFile=c:\firmware\exmp.mtc
- csadm Dev=TCP:288@154.54.2.23 AuthRSASign=ADMIN,:cs2:cp8:COM2 SetTime=20020602111532

Every command running on CryptoServer or CryptoServer LAN requires the 'Dev=' parameter (device parameter, see also 4.4) which sets CryptoServer's or CryptoServer LAN's address. Commands running locally without using a CryptoServer (like module preparation commands, see 4.5) do not need this parameter. Possible values are:

Address	Description
PCI:/dev/cryptoserver0	local CryptoServer No. 1 in a Linux system.
PCI:1	local CryptoServer No. 2 in a Windows system.
TCP:288@194.168.4.107	IP-Address and port number of a CryptoServer LAN
TCP:194.168.4.107	IP-Address of a CryptoServer LAN (default: port=288)
194.168.4.107	IP-Address of a CryptoServer LAN (default: protocol=TCP, port=288)
TCP:288@cslan01	host name and port number of a CryptoServer LAN (using DNS request to resolve host name)
TCP:cslan01	host name of a CryptoServer LAN (using DNS request to resolve host name, default: port=288)
cslan01	host name of a CryptoServer LAN (using DNS request to resolve host name, default: protocol=TCP, port=288)
TCP:3001@127.0.0.1	local CryptoServer SDK.

If the environment variable CRYPTOSERVER is set according to the above mentioned syntax, the 'Dev=' parameter can be skipped (see also 4.1.1).
Using the 'Dev=' parameter overrides the CRYPTOSERVER environment variable in any case for the specific command

4.1.3 Key Specifiers

Some of the CSADM commands use a private or public RSA or ECDSA key to sign a command or a file or to verify a signature. CSADM can handle these keys in three different ways:

(1) RSA/ECDSA keys stored in a file '`*.key`' (as plain text).

(2) RSA/ECDSA keys stored in an encrypted key file '`*.key`' (private key part stored encrypted, public key part stored as plain text).

(3) RSA/ECDSA keys stored on a smart card.

If a command needs a public key, it can be read from a file or from a smart card. If a command needs a private key, it can either be read from a file, or a smart card can be used to calculate the signature. In the latter case the key will not be read out of the smart card. A PIN has to be entered via the PIN-Pad to enable the smart card to generate signatures.

For security reasons private RSA or ECDSA keys should normally be used only from smart cards or encrypted key files. In a test environment, where the private key does not need to be kept secret, it may be useful to store the keys in (plaintext) files.

Encrypted RSA key files are protected by a 168 bit Triple-DES key that is derived from a password with the SHA-256 hashing algorithm. Encrypted ECDSA key files are protected by a 256 bit AES key that is derived from a password with the SHA-256 hashing algorithm. In both cases the password can be changed with the *ChangePassword* command (see 4.13.1). With the same command a plaintext key file can be changed in an encrypted key file and vice versa (by omitting the old respectively the new password).

Apart from test environments, it is strongly recommended to store private keys only on smart cards! Only in this case the private key will never leave the secure token and not be used for calculations on the host.

*For all commands that use RSA or ECDSA keys a **key specifier** has to be given in the command syntax. A key specifier is either*

1. *a filename of a key file ('`*.key`') or*
2. *a smart card specifier.*

Key File Specifiers:

In case that the private part of the key is needed for the command and is given in an encrypted key file, the password of the key file has to be given in the command syntax, too. This can be done either by giving a separate password parameter (Example 1) or by appending the password directly after the filename, separated by a '#' (Example 2):

Example 1: **csadm Password=silence InitPrvKey=c:\keys\init_prv.key BLResetAlarm**

Example 2: **csadm InitPrvKey=c:\keys\init_prv.key#silence BLResetAlarm**

For some commands only the second syntax variant is possible (e.g. if several key specifiers are involved), as described in the relevant command syntax description.

In both cases hidden password entry can be used (see 4.1.4):

Example 1: **csadm Password=ask InitPrvKey=c:\keys\init_prv.key BLResetAlarm**

Example 2: **csadm InitPrvKey=c:\keys\init_prv.key#ask BLResetAlarm**

Smart Card Specifiers:

A smart card specifier always starts with a colon and consists of 3 strings separated by colons (e. g. `:cs2:cp8:COM1`):

1. The first string identifies the type of the smart card.
2. The second string identifies the type of the smart card reader.
3. The last string is the name of the serial device the reader is connected to.

Smart card types supported at the moment are the following types:

Identifier	Smart card type
`cs2`	CryptoServer smart card (using TCOS)
`usa`	Utimaco PKI card (using TCOS)
`cos`	CardOS smart card
`nkey`	Telesec NetKey Card

At the moment supported reader types are:

Identifier	Smart card reader type
`cp8`	Ingenico/Bull SafePad
`cm8`	Omnikey CardMan 8630
`acr`	Advanced Card System ACR80

Key specifier examples:

Key specifier	Description
`C:\my_keys\initprv.key`	Key file.
`:cs2:cp8:COM1`	Key from a CryptoServer smart card using an Ingenico SafePad connected to `COM1` of a Windows PC.
`:cos:cm8:/dev/ttyS0`	Key from a CardOS smart card using a CardMan 8630 reader connected to `ttyS0` of a Unix machine.

4.1.4 Password Entry

To authenticate a security-relevant administration command, an operator who uses a password-based authentication mechanism (*Cleartext Password Authentication, SHA-1-Hashed Password Authentication* or *HMAC Password Authentication*, see 2.6.3) has to enter his user name and password to the CSADM tool (see 4.10). At this and other opportunities it is possible to read the password on the monitor which is connected to the PC on which CSADM is running.

To avoid the password being reflected as plaintext on the monitor, the CryptoServer offers the possibility for hidden password entry.

For hidden password entry, instead of the password first the string ‘ask’ has to be entered. Then CSADM will, before starting to process the authentication (and the rest of the command), return and prompt for the password separately.

Example:

csadm AuthSHA1Pwd=paul,ask SetTime=SYSTEM

Enter Passphrase:

If now the password will be entered over the keyboard, it will not be reflected in clear on the monitor, but be hidden by the display of default characters.

The hidden password entry mechanism can also be used to protect the password of an encrypted key file (see 4.1.3) or the root password of the CryptoServer LAN from being displayed on the monitor.

The usage of hidden password entry is strongly recommended!

4.2 Command Execution with the CSADM Tool

If the CryptoServer is installed on the local computer (as PCI card) commands will be sent from the host to the CryptoServer via the PCI interface. The CryptoServer processes the command and sends the answer (answer data or error code) back to the host.

If the CryptoServer is part of a CryptoServer LAN (CSLAN), commands will be sent from the host to the CryptoServer via a TCP connection. Generally, the TCP server ('daemon') running on the CryptoServer LAN (*csxlan*) forwards incoming commands to (one of) the integrated CryptoServer(s), but a few commands are responded by the CryptoServer LAN itself (e. g. setting of the TCP timeout).

The following illustration shows how commands are executed on a local CryptoServer PCI or a remote CryptoServer LAN:

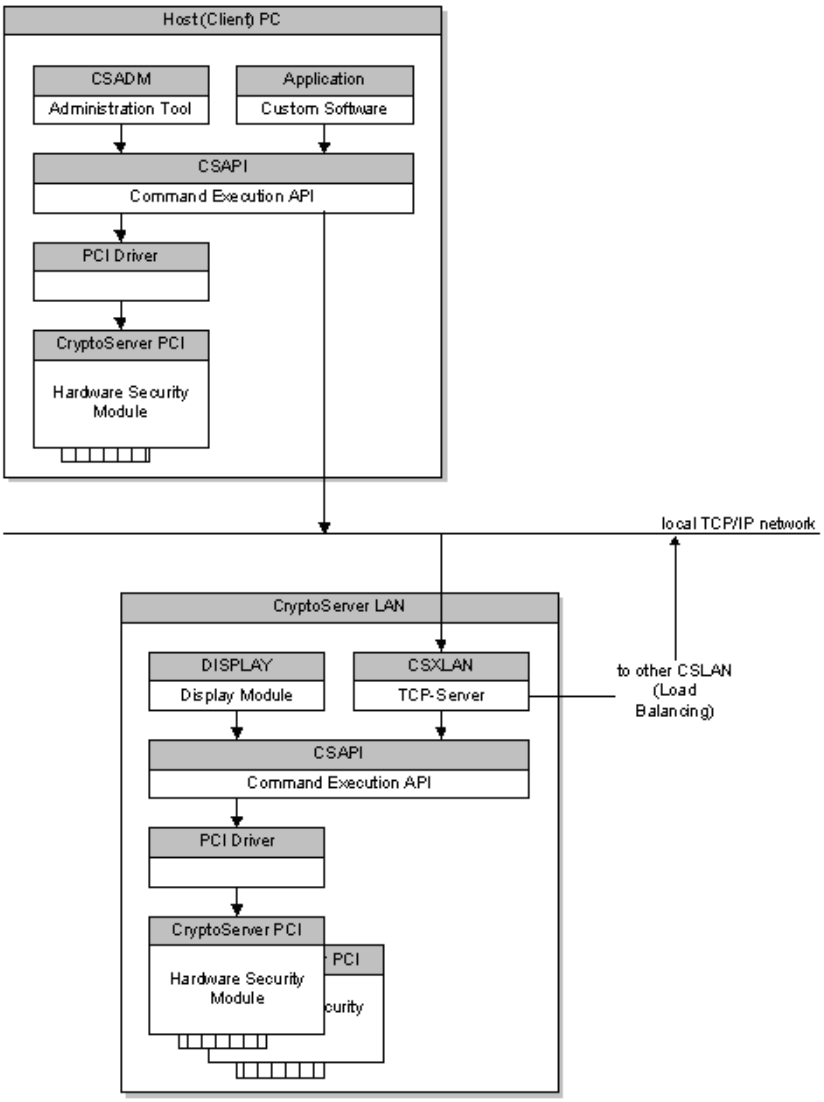


Illustration 4-1: Command Execution

An application on the host PC can use the *command execution library* CSAPI to execute any command on a CryptoServer PCI or LAN.

As a standard application the CryptoServer *Administration Tool* CSADM is available to provide any kind of basic administration like file download or deletion, setting of the CryptoServer's clock, user management a. s. f (see the following subsections).

From the user's point of view it makes no difference whether he accesses a local CryptoServer PCI or a remote CryptoServer LAN. The Administration Tool CSADM is able to send commands either to the local CryptoServer or to the remote CryptoServer LAN (by choosing the appropriate device specifier).

Commands can be divided into the following classes:

Command Destination	**Counterpart on the CryptoServer**	**Required state of the CryptoServer**	**Command Group**
CryptoServer	Boot Loader	*initialized* (lower states are only accessible by Utimaco Safeware AG)	base (initial) administration
	Command Scheduler Module (CMDS)	*operational*	authentication, user management
	Administration Module (ADM)	*operational*	extended administration
	other firmware modules	*operational*	project specific
TCP Server of the CryptoServer LAN	Control module of the TCP Server (daemon) *csxlan*	any	administration (configuration) of the TCP Server ('daemon') *csxlan*

In the following chapters all CSADM commands will be described in detail.

4.3 Basic Commands

These basic commands are CSADM internal functions.

No connection to a CryptoServer will be established.

4.3.1 Help

If called without any parameter, this command shows a list of all available CSADM commands. If a command name is given as a parameter, specific help will be provided.

Syntax	csadm Help csadm Help=<command>
Parameter	<command> specific command of the CSADM
Example	csadm Help=ListFiles
Output	**List File(s) from FLASH / SYS / NVRAM Directory** **syntax:** **csadm ListFiles[=(FLASH\ | SYS\ | NVRAM\)<pattern>]**

4.3.2 PrintError

This command displays the corresponding error message text to an error code. CSADM has a build-in list with all standard error messages of the CryptoServer, CryptoServer LAN, PCI-Driver and Host-API (CSAPI). Error messages for special customer application software are not included in this list and therefore not displayed.

Syntax	csadm PrintError=<errorcode>
Parameter	<errorcode> error code (hexadecimal).
Example	csadm PrintError=B901306F
Output	**Error B901306F** **CryptoServer API LINUX** **can't get connection** **errno = 111**

4.3.3 Version

This command shows the version number of the CSADM.

Syntax	csadm Version
Output	**CryptoServer Administration Utility Ver. 1.0.5**

4.4 Commands to Set-Up Parameters

Most commands require one or more parameters which have to be set up prior to the command execution (and will be evaluated during command execution).

The following table shows all available parameters:

Command	Parameter	Used e. g. by Command(s)
`Dev=` `Device=`	address of CryptoServer or CryptoServer LAN (see 4.1.2)	nearly all
`MTCSignKey=`	key specifier of private key (see 4.1.3).	`MakeMTC`
`MMCSignKey=`	key specifier of private key.	`MakeMTC`
`MTCPubKey=`	key specifier of public key.	`VerifyMTC`
`MMCPubKey=`	key specifier of public key.	`VerifyMTC`
`SignType=`	specifier for signature format: ■ '1' for signature calculation according to PKCS#1 [PKCS#1] ■ '2' for signature calculation according to proprietary format	`MakeMTC`
`InitPubKey=`	key specifier of public key.	`BLChangeInitKey`
`InitPrvKey=`	key specifier of private key.	many boot loader commands
`InitBckKey=`	key specifier of XOR-halves of key on back-up smartcards	`SaveKey`
`MdlSigPubKey=`	key specifier of public key.	`BLLoadInitKey`
`Password=`	password of encrypted key file	many commands, like e. g. `MakeMTC`, `BLClear`, `AuthRSASign`, `SessionRSA`, `ChangePassword, ...`
`NewPassword=`	new password of encrypted key file	`ChangePassword`, `GenKey`
`Admin3=`	string of up to 16 characters.	`BLChangeInitKey`
`Pause=`	"1" or "0", to switch on or off	
`UserKey=`	user name and key specifier for user's authentication key	`MBKGenerateKey`, `MBKImportKey`
`Key=`	key specifier for MBK key shares	`MBKGenerateKey`, `MBKImportKey`, `MBKCopyKey`

Command	Parameter	Used e. g. by Command(s)
`FWEncKey=`	key specifier for firmware encryption key (public RSA key).	`MakeMTC`
`FWDecKey=`	key specifier for firmware decryption key (private RSA key).	`SaveKey`
`OldKey=`	key specifier of old MBK	`MBKImportKey`

If the environment variable CRYPTOSERVER is set, the 'Dev=' parameter can be skipped (see also 4.1.1).
Using the 'Dev=' parameter overrides the CRYPTOSERVER environment variable in any case for the specific command.

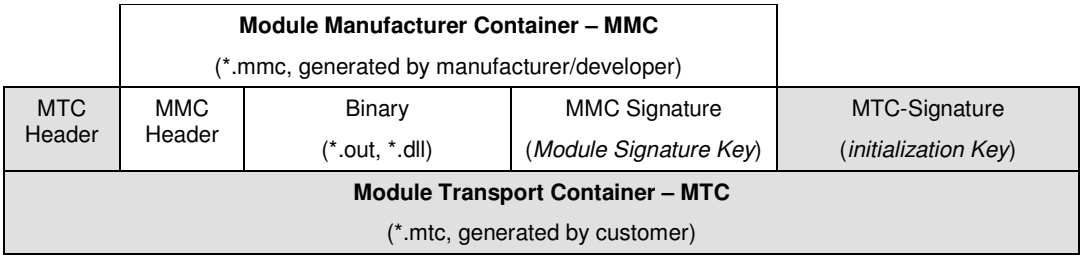
4.5 Commands to Prepare Firmware Modules

These commands are used for the load preparation of firmware modules, in particular the making, verification and removing of the *Module Transport Container* (MTC) or the *Module Manufacturer Container* (MMC).

Furthermore commands are offered to create a firmware package file "*.mpkg" or to see its contents, or to retrieve the contained modules from a "*.mpkg" file.

More details about the concepts of MTCs, MMCs and package files can also be found in chapter 3.7.

All firmware modules are packed into two (data) containers before they are loaded into the CryptoServer:



- The – inner - Module Manufacturer Container (MMC):

  Signing the *Module Manufacturer Container* is mandatory, the private part of the manufacturer's *Module Signature Key* will have to be used for its calculation. The signature will be checked by the operating system (SMOS) of the CryptoServer every time the firmware module is started.

  The MMC is intended to be created by the author of the firmware module s. t. the CryptoServer - and anyone else - is able to check the authenticity of the firmware module's origin.

- The – outer - Module Transport Container (MTC):

  Signing this container is mandatory. This signature has to be calculated with the private part of the *Initialization Key* and will be checked by the operating system (SMOS) of the CryptoServer every time the firmware module is started.

  The MTC is intended to be created by the user of the CryptoServer (customer) with the aim of personalizing the firmware module. Since the CryptoServer does not start firmware modules without correct signatures, manipulation of the firmware module during storage or transportation is impossible.

The structure and signature of both containers of a firmware module can be checked manually (outside the CryptoServer) using the command *VerifyMTC* (see chapter 4.5.3). For more details about these containers, their purpose and their usage, see 3.7.1.

It is neither necessary nor possible for the customer to remove and rebuild the inner MMC since only the manufacturer is able to sign this container.
The outer MTC of all firmware modules has to be removed and rebuilt if the CryptoServer's Initialization Key *has been changed (see* BLChangeInitKey *command, chapter 4.7.5).*

The following command group contains internal functions of the CSADM. No connection to a CryptoServer will be established.

4.5.1 MakeMTC

This command builds the *Module Transport Container* file (*.mtc) from a *Module Manufacturer Container* file (*.mmc) or a raw firmware module (*.out, *.dll). The signature of the MTC has to be done with the private part of CryptoServer's *Initialization Key* which has to be entered as the *MTCSignKey* key specifier.

If a raw (binary) firmware module (*.out or *.dll) is given as input file, in a first step the MMC file (*.mmc) will be built: To do this, the private part of the *Module Signature Key* has to be given as parameter *MMCSignKey*.

Since the public part of the Module Signature Key*, used to verify the MMC during the starting phase of the firmware module, was loaded onto the CryptoServer by the manufacturer during the production phase, only the manufacturer is able to build a signed – and working – MMC.*

Syntax	csadm [MMCSignKey=<keyspec>] [Password=<password>] MTCSignKey=<keyspec> [FWEncKey=<keyspec>] [SignType=<signaturetype>] MakeMTC=<file>		
Parameters	<keyspec>	MMCSignKey:	private part of the *Module Signature Key*
		MTCSignKey:	private part of the *Initialization Key*
		FWEncKey	public part of firmware encryption key
	(see 4.1.3 for possible key specifiers)		
	<password>	if *Initialization Key* is given in encrypted key file: password of that encrypted key file *(see 4.1.3 for encrypted key files)*: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: password of key file	
	<signaturetype>	'1' / '2' (see below)	
	<file>	firmware module (*.mmc, *.out, *.dll)	

Example	csadm MTCSignKey=c:\keys\init_prv.key MakeMTC=c:\firmware\exmp.mmc
Output	**building MTC ... OK** on success, or error message

Note	■ Signing the MMC will only be done if a MMCSignKey is given. Since the private part of the *Module Signature Key* is needed to create the signature, only the manufacturer of the CryptoServer is able to build a signed (and working) MMC.
	■ Signing the MTC is mandatory. The signature has to be created using the private part of the *Initialization Key* as MTCSignKey. The corresponding public part of the *Initialization Key* has to be loaded onto the CryptoServer.
	■ If the *Initialization Key* is given in an encrypted key file, entering the respective key file's password is mandatory.
	■ With the optional parameter "SignType" it can be chosen according to which format/standard the MTC signature will be calculated:
	  ■ Setting the parameter 'SignType=1' means that the MTC signature will be calculated according to PKCS#1 [PKCS#1].
	  ■ Setting the parameter 'SignType=2' means that the MTC signature will be calculated according to some CryptoServer proprietary signature format.
	In case the SignType parameter is omitted, the default value 'SignType=2' will be used.
	■ If the MTCSignKey (or MMCSignKey) is stored on a smart card, the card has to be inserted into the smart card reader on demand and the PIN has to be entered.
	■ If FWEncKey is given, then raw firmware module is encrypted with that key.
	■ If the command has successfully been performed the signed firmware module is stored in the actual directory. If a raw firmware module had been given as input (*.out, *.dll), the interim MMC file will be deleted.

Should multiple MTCs be built in one step, the last part of the command (MakeMTC=<file>) will have to be repeated for each firmware module.

4.5.2 RemoveMTC

This command removes the header and signature of a given container (MTC or MMC):

If a *Module Transport Container* (*.mtc) is given, the command removes the MTC header and signature and restores the contained *Module Manufacturer Container* (*.mmc).

If a *Module Manufacturer Container* (*.mmc) is already given as input file its header and signature is removed too and the contained binary firmware module file (*.out or *.dll) is restored. Since a valid MMC signature can only be generated by Utimaco, it should not be removed by the customer.

Syntax	csadm RemoveMTC=<file>
Parameter	<file> firmware module as MTC (*.mtc), or firmware module as MMC (*.mmc)
Example	csadm RemoveMTC=c:\firmware\exmp.mtc
Output	**removing MTC ... OK** on success, or error message
Note	The unpacked file will be stored in the current directory.

Should multiple MTCs be removed in one step, the last part of the command (RemoveMTC=<file>) will have to be repeated for each firmware module.

4.5.3 VerifyMTC

This command verifies the signature of the *Module Transport Containers* MTC (*.mtc) and the signature of the *Module Manufacturer Container* MMC (*.mmc).

Syntax	csadm MMCPubKey=<keyspec> MTCPubKey=<keyspec> VerifyMTC=<file>
Parameter	<keyspec> MMCPubKey: public part of the *Module Signature Key* MTCPubKey: public part of the *Initialization Key* *(see 4.1.3 for possible key specifiers)* <file> firmware module as MTC (*.mtc), or firmware module as MMC (*.mmc)
Example	csadm MMCPubKey=:cs2:cp8:COM1 MTCPubKey=c:\keys\init_pub.key ... ... VerifyMTC=c:\firmware\exmp.mtc
Output	**Verifying MTC ...OK** **Removing MTC ...OK** **Verifying MMC ...OK**
Note	While verifying a MTC, CSADM is creating a temporary MMC file in the current directory.

Should multiple MTCs be verified in one step, the last part of the command (VerifyMTC=<file>) will have to be repeated for each firmware module

4.5.4 ModuleInfo

This command shows the module information of a firmware module. MTC, MMC or raw binary files can be given as input file. If the input file is a MTC, the name of the signing key (usually the private part of *Initialization Key*) is displayed, too.

Syntax	csadm ModuleInfo=<file>		
Parameter	<file> firmware module (*.mtc, *.mmc, *.out, *.dll)		
Example	csadm ModuleInfo=adm.mtc		
Output	**Module**	**'ADM'**	// module name
	ID	**87**	// module ID (FC)
	Version	**1.0.0.3**	// module version number
	Name	**'Administration Module'**	// extended module name
	Signature Info	**Utimaco Safeware AG / Init-Dev-1-Key**	// owner and name of the MTC signing key (*Initialization Key*)
Note			

Should multiple firmware modules be processed in one step, the last part of the command (ModuleInfo=<file>) will have to be repeated for each firmware module

4.5.5 RenameToVersion

This command expands the filename of a firmware module by inserting the version number of the firmware module.

On loading onto the CryptoServer this added version number will be automatically removed.

Syntax	csadm RenameToVersion=<file>
Parameter	<file> firmware module (*.mtc, *.mmc, *.out, *.dll)
Example	csadm RenameToVersion=C:\modules\vdes.mtc
Output	none on success, or error message
Note	In the example above, the file `C:\modules\vdes.mtc` is renamed to e. g. `C:\modules\vdes_1.0.1.0.mtc`

4.5.6 Pack

This command creates a CryptoServer package file (archive "*.mpkg", see 3.7.2) from the content of the given directory. All files contained in the given directory will be added to the package file (which has the same name as the given directory, plus extension ".mpkg"). This package file can later be used to load a set of several firmware modules and files into the CryptoServer within one command (*LoadPkg* command, see 4.8.15).

Syntax	csadm Pack=<directory>
Parameter	<directory> Input directory containing CryptoServer firmware modules and files
Example	csadm Pack=firmware
Output	none on success, or error message
Note	The resulting package file "firmware.mpkg" will be stored in the same directory as the given input directory 'firmware'. Any existing file with the same name will be overwritten.

4.5.7 Unpack

This command extracts all files contained in a CryptoServer package file "*.mpkg" (see 3.7.2). The files will be extracted in a new directory which will have the same name as the given package file but without the extension ".mpkg".

Syntax	csadm Unpack=<package>
Parameter	<package> Input package file containing CryptoServer firmware modules and files (filename must have extension ".mpkg")
Example	csadm Unpack=firmware.mpkg
Output	none on success, or error message
Note	The resulting directory 'firmware' containing the extracted firmware modules and files will be created in the same directory where the given package file "firmware.mpkg" is located. If a directory of that name already exists, the extraction is denied.

4.5.8 ListPkg

This command lists the content of a CryptoServer package file (archive "*.mpkg", see 3.7.2), i. e. the filenames of all contained files. If firmware modules are contained in the package the module's version numbers and long names are also listed.

Syntax	csadm ListPkg=<package>
Parameter	<package> Input package file containing CryptoServer firmware modules and files (filename must have the extension ".mpkg")
Example	csadm ListPkg=firmware.mpkg
Output	Example output: **Archive firmware.mpkg contains:** **1. - adm_1.0.1.1.mtc (Administration Module version 1.0.1.1)** **2. - asn1_1.0.1.2.mtc (ASN1 Module version 1.0.1.2)** **3 - cmds_1.0.6.0.mtc (Command Scheduler version 1.0.6.0)** **4. - db_1.0.1.1.mtc (Database module version 1.0.1.1)** **5. - hash_1.0.1.0.mtc (Hash Module version 1.0.1.0)** **6. - lna_1.0.3.0.mtc (Long Number Arithmetic version 1.0.3.0)** **7. - smos_1.0.3.5.mtc (SMOS Operating System version 1.0.3.5)** **8. - util_1.0.5.0.mtc (Utility Module version 1.0.5.0)** **9. - vdes_1.0.0.3.mtc (DES Module version 1.0.0.3)** **10. - vrsa_1.0.4.0.mtc (RSA Module version 1.0.4.0)**

4.5.9 VerifyPkg

This command verifies the signature of the *Module Transport Container* MTC (*.mtc) and (optionally) the signature of the *Module Manufacturer Container* MMC (*.mmc) of each firmware module that is contained in the given package file (archive "*.mpkg", see 3.7.2).

This command can be performed without any access to a CryptoServer.

Syntax	csadm [MMCPubKey=<keyspec>] MTCPubKey=<keyspec> VerifyPkg=<package>		
Parameters	<keyspec>	MMCPubKey:	public part of the *Module Signature Key*
		MTCPubKey:	public part of the *Initialization Key*
	(see 4.1.3 for possible key specifiers)		
	<package>	input package file containing CryptoServer firmware modules and files (filename must have extension ".mpkg")	
Example	csadm MMCPubKey=:cs2:cp8:COM1 MTCPubKey=c:\keys\init_pub.key VerifyPkg=c:\firmware\fw-1.0.0.mpkg		
Output	Information about the MMC and MTC signatures (verification results, see example below).		

Example:

```
   csadm MMCPubKey=:cs2:cp8:COM1 MTCPubKey=c:\keys\init_pub.key
       VerifyPkg=c:\firmware\fw-1.0.0.mpkg

I: Verifying db_1.1.0.0_c64.mtc        MTC OK - MMC OK
I: Verifying vdes_1.0.1.1_c64.mtc      MTC OK - MMC OK
I: Verifying aes_1.0.3.0_c64.mtc       MTC OK - MMC OK
I: Verifying asn1_1.0.3.3_c64.mtc      MTC OK - MMC OK
I: Verifying smos_2.0.0.6_c64.mtc      MTC OK - MMC OK
I: Verifying hash_1.0.5.0_c64.mtc      MTC OK - MMC OK
I: Verifying eca_1.1.0.4_c64.mtc       MTC OK - MMC OK
I: Verifying ecdsa_1.0.1.0_c64.mtc     MTC OK - MMC OK
I: Verifying dsa_1.0.0.0_c64.mtc       MTC OK - MMC OK
I: Verifying util_2.1.0.0_c64.mtc      MTC OK - MMC OK
I: Verifying adm_2.0.0.3_c64.mtc       MTC OK - MMC OK
I: Verifying cmds_1.1.1.0_c64.mtc      MTC OK - MMC OK
I: Verifying pkcs11_1.0.5.1_c64.mtc    MTC OK - MMC OK
I: Verifying vrsa_1.0.6.0_c64.mtc      MTC OK - MMC OK
I: Verifying lna_1.0.4.3_c64.mtc       MTC OK - MMC OK

Package c:\firmware\fw-1.0.0.mpkg successfully verified
```

4.5.10 ResignPkg

This command recalculates the signature of the *Module Transport Container* MTC (*.mtc) of each firmware module contained in the given package file (archive "*.mpkg", see 3.7.2). To get a downloadable package file, the new MTC signatures have to be calculated with the CryptoServer's *Initialization Key*.

This command can be performed without any access to a CryptoServer.

Syntax	csadm [Password=<password>] MTCSignKey=<keyspec> ResignPkg=<package>,<outfile>
Parameters	<keyspec> MTCSignKey: private part of the *Initialization Key* *(see 4.1.3 for possible key specifiers)* <password> if *Initialization Key* is given in encrypted key file: password of that encrypted key file *(see 4.1.3 for encrypted key files)*: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: password of key file <package> Input package file containing CryptoServer firmware modules and files (filename must have extension ".mpkg") <outfile> Name of the output package file containing re-signed modules. If the extension ".mpkg" is missing it will be automatically appended.
Example	csadm MTCSignKey=:cs2:cp8:COM1 ResignPkg=c:\firmware\fw-1.0.0.mpkg,c:\firmware\fw_sig_new-1.0.0.mpkg
Output	Status information (see example below).

Example:

```
csadm MTCSignKey=:cs2:cp8:COM1
  ResignPkg=c:\firmware\fw-1.0.0.mpkg, c:\firmware\fw_sig_new-1.0.0

Package c:\firmware\fw-1.0.0.mpkg successfully resigned

Created archive is c:\firmware\fw_sig_new-1.0.0.mpkg
```

4.6 CryptoServer Driver Commands

The commands in this section are directed to the PCI driver of the CryptoServer. If the addressed CryptoServer is part of a CryptoServer LAN, these commands are executed remotely via TCP.

The CryptoServer may be in any state (see 3.2) to execute the driver commands. No authentication towards the CryptoServer is required.

If the addressed CryptoServer is part of a CryptoServer LAN it might be necessary to authenticate the driver commands Reset, Restart *and* ResetToBL *towards the CryptoServer LAN with the root password of the LAN box.*

This depends on the configuration of the LAN box: if the 'AuthReset' variable is set in the configuration file, password authentication of these commands is mandatory (see [CSLANUserManual] section 5.2.5 *The Configuration File csxlan.conf*). This password protection can be useful e. g. to avoid a denial-of-service-attack.

4.6.1 Reset

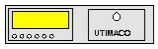
This command performs a hardware reset (like Power Off-On) of the CryptoServer on PCI level.

After executing *Reset* it takes up to 15 seconds until the boot loader window is timed out and the operating system is restarted again (see chapter 2.3.3).

It is therefore recommended NOT to use the Reset *command but instead ...*

- *to use Restart (see 4.6.3) to restart the CryptoServer as fast as possible (3 – 5 sec) or*
- *to use ResetToBL (see 4.6.2) to reset the CryptoServer and to get into boot loader mode.*

Syntax	csadm [Dev=<device>] Reset[=<password>]
Parameters	<device> device specifier (see 4.1.2) <password> root password of CryptoServer LAN (see below): ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password This parameter is only relevant if the CryptoServer LAN is used, and if the command is password protected (see below).
Output	none on success, or error message
Note	In case that the CryptoServer LAN is used and the driver commands *Reset*, *Restart* and *ResetToBL* are protected by a root password (i. e. depending on the settings in the LAN box configuration file), the 'password' parameter is mandatory. See page 92 above.



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Generic Commands` ➔ `Reset CryptoServer`

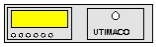
4.6.2 ResetToBL

This command will reset the CryptoServer and get it into *boot loader mode* (see 2.3.4):

First a reset of the CryptoServer is performed in the same way as using the *Reset* command. Immediately after the *Reset* command a dummy command is sent to the CryptoServer. The boot loader now remains active and will not start the operating system SMOS. Thus the CryptoServer is in boot loader mode now.

- *If the CryptoServer is in boot loader mode, the operating system (SMOS) and all other firmware modules can be started - and the CryptoServer therefore can be put into operational mode again - using the StartOS command (see 4.7.2).*
- *The RecoverOS command (see 4.7.3) on the other hand starts the recovery copies of the firmware modules, which were loaded into the CryptoServer during the initial set-up.*
- *In boot loader mode no application (no other firmware modules) is running!*

Syntax	csadm [Dev=<device>] ResetToBL[=<password>]
Parameters	<device> device specifier (see 4.1.2) <password> root password of CryptoServer LAN (see below): ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password This parameter is only relevant if the CryptoServer LAN is used, and if the command is password protected (see below).
Output	none on success, or error message
Note	■ In boot loader mode any command sent to the CryptoServer is responded by the boot loader. ■ The boot loader will remain active until the CryptoServer is reset again or the *StartOS*-command is sent to the CryptoServer. ■ In case that the CryptoServer LAN is used and the driver commands *Reset*, *Restart* and *ResetToBL* are protected by a root password (i. e. depending on the settings in the LAN box configuration file), the 'password' parameter is mandatory. See page 92 above.



With CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration ➔ Generic Commands ➔ Reset CryptoServer to bootloader`

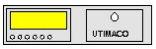
4.6.3 Restart

This command will reset/restart the CryptoServer and get it into *operational mode* (see 2.3.4):

In a first step a reset of the CryptoServer is performed in the same way as using the *Reset* command. In addition to the *Reset* command the *StartOS* command is sent to the CryptoServer immediately.

Using the Restart *command is the fastest way to restart the operating system. If it is successful, the CryptoServer is in operational mode afterwards.*

Syntax	csadm [Dev=<device>] Restart[=<password>]
Parameters	<device> device specifier (see 4.1.2) <password> root password of CryptoServer LAN (see below): ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password This parameter is only relevant if the CryptoServer LAN is used, and if the command is password protected (see below).
Output	none on success, or error message
Note	■ As a general rule, the CryptoServer has to be restarted after the loading (or replacement) of a new firmware module: The firmware module only becomes active after the CryptoServer having been restarted. ■ In case that the CryptoServer LAN is used and the driver commands *Reset*, *Restart* and *ResetToBL* are protected by a root password (i. e. depending on the settings in the LAN box configuration file), the 'password' parameter is mandatory. See page 92 above.



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Generic Commands` ➔ `Restart CryptoServer`

4.6.4 GetInfo

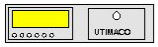
This command retrieves information from the PCI driver of the CryptoServer. It shows the driver version, PCI slot number, interrupt number, battery state, time-out, state of the error correction and the contents of the PCI registers.

The GetInfo *command will be executed even if the CryptoServer does not respond to any command (even the* GetState *command). In some cases the information provided this way helps to identify CryptoServer's low level problems. The interpretation of the output requests extensive knowledge of the CryptoServer and is therefore not explained in more detail.*
Please prepare this output in case of a support request!

Syntax	csadm [Dev=<device>] GetInfo
Parameter	<device> device specifier (see 4.1.2)
Output	**vers 1.0.4.0** **slot 11** **irq 10** **batt ok** **timeout 600000** **tx idle** **rx idle** **txrt 0 0** **rxrt 0 0** **mbr 00100021 00000020** **bar0 00000000 00ABF000 00 Master Write Address0** **bar0 00000000 00000020 08 MW Count Status0 / MW Transfer Count0** **bar0 4B525950 544F4B4F 10 Master Write Address1** **bar0 00000000 00000000 18 MW Count Status1 / MW Transfer Count1** **bar0 00000000 000F0100 20 misc** **bar0 0E040000 00000000 28 misc** **bar0 00000008 00000000 30 I2O** **bar0 00000000 00000000 38 reserved** **bar0 FFFFFFFF FFFFFFFF 40 I2O** **bar0 00000000 00AC2000 48 Master Read Address0 / Chain Desc. Start Address0** **bar0 00000000 00000010 50 Master Read Count Status0 / Master Read Transfer Count0** **bar0 00000000 00000000 58 Master Read Address1 / Chain Desc. Start Address1** **bar0 00000000 00000000 60 Master Read Count Status1 / Master Read Transfer Count1** **bar0 08080808 08080808 68 misc** **bar0 00400020 00000020 70 PCI Outgoing MailBox Register Host->CS2** **bar0 00100021 000000XX 78 PCI Incomming MailBox Register CS2->Host** **...**
Note	The command is used for diagnostic purposes in error case!

The GetInfo *command has to be regularly used to check the CryptoServer's battery state which is announced with the line* `'batt ...'`.

In case that the battery state is not `'batt ok'`*, one or more of the CryptoServer's batteries have to be exchanged. See 5.1.*



With the CryptoServer LAN, this command will be performed by choosing the menu point

`CryptoServer administration` ➔ `Generic Commands` ➔ `Show driver info`

4.7 Boot Loader Commands for Base Administration

This command group addresses CryptoServer's boot loader, which offers functionality for base (initial) administration.

In boot loader mode (see 2.3.4) security relevant commands have to be signed with the private part of the *Initialization Key* or (partly) *Production Key*. The System Administrator (unnamed in boot loader mode, but later called 'ADMIN'), who is owner of the *Initialization Key*, is responsible for this part of administration.

The following table shows all available boot loader commands and the required mode, state and authentication key of the CryptoServer:

Command	***required CryptoServer mode***	***required CryptoServer state***	***Authentication key***
GetState	boot loader mode or operational mode	any	none
StartOS	boot loader mode	initialized	none
RecoverOS	boot loader mode	initialized	none
ListFiles	boot loader mode or operational mode	initialized or operational	none
BLLoadFile	boot loader mode	initialized	Initialization Key
BLChangeInitKey	boot loader mode	initialized	Initialization Key
BLClear	boot loader mode	initialized	Initialization Key / Production Key
BLSetRTC	boot loader mode	initialized	Initialization Key
		produced	Production Key
BLResetAlarm	boot loader mode	initialized	Initialization Key
		produced	Production Key
GetBootLog	boot loader mode or operational mode	initialized or operational	none
GetAlarmLog	boot loader mode or operational mode	initialized or operational	none
GetTempLog	boot loader mode or operational mode	initialized or operational	none
GetTimeLog	boot loader mode or operational mode	initialized or operational	none
GetTime	boot loader mode or operational mode	initialized or operational	none
LoadPkg	boot loader mode or operational mode	initialized or operational	Initialization Key
CheckPkg	boot loader mode or operational mode	initialized or operational	none
Test	boot loader mode or operational mode	initialized or operational	none

As a special case the commands

- *GetState, ListFiles, GetTime, GetBootLog, GetAlarmLog, GetTempLog, GetTimeLog, Test*
- *LoadPkg*
- *CheckPkg*

will be executed in boot loader mode as well as in operational mode.

The first of these commands, GetState, ListFiles, GetTime, GetBootLog, GetAlarmLog, GetTempLog, GetTimeLog *and* Test*, are responded by the boot loader and the Administration Module (or CMDS module) in the same way.*

The two CSADM commands LoadPkg *and* CheckPkg *take a special position because they do not correspond to exactly one CryptoServer command each but consist of a series of CryptoServer commands, executed one after another, and supplemented by actions of the CSADM itself. Some of these executed CryptoServer commands may address CryptoServer's boot loader, others address CryptoServer's Administration Module. But switching from one mode to another (if necessary) will be done automatically and transparent for the user.*

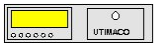
4.7.1 GetState

This command returns the state of the CryptoServer, its temperature, alarm state, hardware and setup information.

The command GetState *is responded by the CryptoServer in any mode (by the boot loader as well as by the administration module) and therefore should be executed as a first diagnostic measure in case of problems.*
Please prepare the output of GetState *in case of a support request!*

Syntax	csadm [Dev=<device>] GetState
Parameter	<device> device specifier (see 4.1.2)
required state	any
Authentication	none
Output	**CryptoServer is in Bootloader Mode** **state = 00000004 INITIALIZED** **temp = 47,5 C** **alarm = OFF** **bl_ver = 01000500** **hw_ver = 01000200** **UID = f4000006 fa1ee701** **adm1 = 5554494d 41434f20 43533030 30303036 | UTIMACO CS000006** **adm2 = 5376656e 27730000 00000000 00000000 | Sven's** **adm3 = 496e6974 2d446576 2d312d4b 65790000 | Init-Dev-1-Key**

For the meaning of the displayed fields see section 4.8.1 where the performance of *GetState* in operational mode is explained (which is identical to the performance in boot loader mode).



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Bootloader functions` ➔ `Get status of CryptoServer`

4.7.2 StartOS

CryptoServer's operating system SMOS (smos.mtc) and other firmware modules (*.mtc) will be started from CryptoServer's working directory (\FLASH). This process will be performed in two steps:

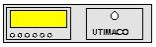
First SMOS will be started by the boot loader from the \FLASH directory. If this was successful, the StartOS command returns and the boot loader terminates; SMOS is now active. Then automatically in a second step SMOS starts in turn all other firmware modules that are present in the \FLASH directory.

See 2.3.3.2 for more detailed information about this process.

On start-up of the CryptoServer (power-on or after the Reset *command) the* StartOS *command is automatically executed by the boot loader if it doesn't receive any command within 10 seconds (see chapter 2.3.3.1)*

Syntax	csadm [Dev=<device>] StartOS
Parameter	<device> device specifier (see 4.1.2)
Authentication	none
Output	none on success, or error message
Note	■ If no operating system SMOS is present (SMOS has not been loaded onto CryptoServer before), an error message will be displayed. ■ On start-up of SMOS and other firmware modules the signatures of all MTC/MMC containers and the integrity of the firmware modules are checked. ■ If an error occurs during the start-up of a firmware module, the corresponding error message will be added to the boot log file (see *GetBootLog*, section 4.8.8).

This command can only be performed in boot loader mode and if the CryptoServer is in the initialized *state. After it is successfully performed, the CryptoServer is in the* operational *state and works in (normal) operational mode (see 2.3.4)*



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Bootloader functions` ➔ `Start OS (normal mode)`

4.7.3 RecoverOS

The back-up copies of the operating system SMOS and any other basic firmware modules (*.mtc) that are found in CryptoServer's system directory (\SYS) will be started. This process will be performed in two steps:

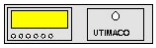
First SMOS will be started by the boot loader from the \SYS directory. If this has been successfully completed, the *RecoverOS* command returns and the boot loader terminates; SMOS is now active. Then automatically in a second step SMOS itself starts all other firmware modules that are present in the \SYS directory. See 2.3.3.5 for more detailed information.

Due to the size limitation of the \SYS directory (approximately 400 Kbytes free space) only a few firmware modules can be stored there during the first initial set-up of the CryptoServer (see *BLLoadFile*, chapter 4.7.6).

The back-up copies of the firmware modules from the system directory have to be started explicitly by the user, using this RecoverOS *command.*
This may be necessary if it is not longer possible to start the regular set of firmware modules (which are stored in the working directory \FLASH), for example because one or more indispensable modules (SMOS, CMDS, ADM, UTIL) have been deleted by mistake, or a defect firmware module has been loaded during the development phase

Syntax	csadm [Dev=<device>] RecoverOS
Parameter	<device> device specifier (see 4.1.2)
Authentication	none
Output	none on success or error message
Note	■ If the operating system SMOS has not been loaded onto the CryptoServer before, an error message is displayed. ■ On start-up of SMOS and the base firmware modules the signatures of all MTC/MMC containers and the integrity of the firmware modules are checked. ■ If an error occurs during start-up of any firmware module the corresponding error message will be added to the boot log file (see *GetBootLog*, section 4.8.8).

This command can only be performed in boot loader mode and if the CryptoServer is in the initialized *state. After it is successfully performed, the CryptoServer is in the* operational *state and works in (failsafe) operational mode (see 2.3.4)*



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration ➔ Bootloader functions ➔ Start OS (recovery mode)`

4.7.4 BLClear

This command erases the CryptoServer to the *initialized* or *produced* state (see also 3.5).

The following table shows the required authentication keys and the erased data inside the CryptoServer:

Clear to ...	***required authentication key***	***erased data***
Produced	*Production Key*	■ all data in the SYS and FLASH directory without: ▣ the boot loader code ▣ the configuration file 'bl.ini' ▣ the Production Key ▣ the file 'alarm.log' ▣ the field adm1 of the EID ■ the Master Key $\mathbf{K_{CS2}}$
Initialized	*Initialization Key*	■ all data in the SYS and FLASH directory without: ▣ the boot loader code ▣ the configuration file 'bl.ini' ▣ the Production Key ▣ the Module Signature Key ▣ the Initialization Key ▣ the file 'alarm.log' ▣ the field adm1 of the EID ▣ the field adm2 of the EID ■ the Master Key $\mathbf{K_{CS2}}$

Clear-to-Initialized *should be executed …*

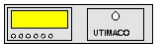
- *before or after changing the Initialization Key (e. g. when switching from test to live operation).*
- *if the back-up copies of the base firmware modules should be updated.*

Clear-to-Produced *should be executed ...*

- *if the private part of the Initialization Key (e. g. the smart card) has been lost.*

To clear the CryptoServer to the produced *state the private part of the Production Key is needed. Therefore in this case the CryptoServer has to be sent back to Utimaco Safeware AG.*

Syntax	csadm [Dev=<device>] [Password=<password>] [InitPrvKey | ProdPrvKey]=<keyspec> BLClear
Parameter	<device> device specifier (see 4.1.2) <password> if *Initialization Key* respectively *Production Key* is given in encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <keyspec> Clear-to-Initialized: private part of the *Initialization Key* Clear-to-Produced: private part of the *Production Key* *(see 4.1.3 for possible key specifiers)*
Examples	■ csadm InitPrvKey=c:\keys\init_prv.key BLClear ■ csadm ProdPrvKey=c:\keys\prod_prv.key BLClear
required state	*Initialized* (boot loader mode required)
Authentication	*Initialization Key* or *Production Key*
Output	none on success or error message
Note	■ If the *Initialization Key* is given as key specifier the CryptoServer is cleared to the *initialized* state. ■ If the *Production Key* is given as key specifier the CryptoServer is cleared to the *produced* state.



With the CryptoServer LAN, the Clear-to-Initialized *command will be performed by choosing the menu point*
`CryptoServer administration` ➔ `Bootloader functions` ➔ `Clear CryptoServer`
The Clear-to-Produced *command cannot be performed via the CryptoServer LAN menu.*

4.7.5 BLChangeInitKey

With this function the public part of a new *Initialization Key* will be loaded onto the CryptoServer. This new *Initialization Key* will replace (i. e. overwrite) the old one (stored as 'Init.KeyRsaPub' in the system directory \SYS of the flash file).

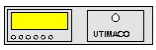
If given, the 'adm3'-field of CryptoServer's Extended Identifier (EID) will be written (see 2.2), otherwise this field will remain empty.

In case of loss or damage of the smart card containing the Initialization Key *of the CryptoServer, base administration using the boot loader is no longer possible for the customer, and the CryptoServer has to be sent back to Utimaco Safeware AG (where a new* Initialization Key *will be loaded, using the* Production Key*).*
Therefore it is highly recommended that

- a *second, identical copy of the smart card containing the Initialization Key is prepared immediately,*
- *these two smart cards are carried by two different persons (System Administrator and his representative),*
- *the 'adm3'-field contains the name of the new Initialization Key (in this way there will never be the question about which key has been loaded as Initialization Key).*

Syntax	csadm [Dev=<device>] [Password=<password>] InitPrvKey=<keyspec> ... [Admin3=<string>] BLChangeInitKey=<keyspec>
Parameters	<device> device specifier (see 4.1.2) <password> if *Initialization Key* is given in encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <keyspec> InitPrvKey: private part of old *Initialization Key* BLChangeInitKey: public part of new *Initialization Key* *(see 4.1.3 for possible key specifiers)* <string> 16 bytes string (should be name of new *Initialization Key*). If given, this string will be stored as 'adm3'-field of CryptoServer's Extended Identifier (EID, see 2.2).
Example	csadm InitPrvKey=c:\keys\init_old_prv.key ... Admin3='Init-Live01-Key' ... BLChangeInitKey=c:\keys\init_new_pub.key
required state	*initialized* (boot loader mode required)
Authentication	old *Initialization Key*
Output	none on success or error message

Note	■ After the *Initialization Key* has been changed, the loaded firmware modules, signed with the old *Initialization Key*, could not longer be started. Therefore, before or after the changing of the *Initialization Key* all data inside the CryptoServer has to be cleared (*BLClear* command, chapter 4.7.4). ■ When the *Initialization Key* has been changed and the CryptoServer has been cleared (see above), the complete set of firmware modules must be re-signed with the new *Initialization Key* (see *RemoveMTC* and *MakeMTC*, chapter 4.5) and again loaded onto the CryptoServer.



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Bootloader functions` ➔ `Change Init Key`

4.7.6 BLLoadFile

The *BLLoadFile* command loads a file onto the CryptoServer. This file is usually a firmware module (**.mtc*) but it is basically also possible to load any other file onto the CryptoServer.

Any file loaded by *BLLoadFile* is stored two-fold: in the working directory (\FLASH) and – as a recovery measure – in the system directory (\SYS) of the flash file, which can only be written by the boot loader.

BLLoadFile does not replace existing files! For replacing an existing file in the working directory (e. g. updating a firmware module with a newer version) the *LoadFile* command has to be used which is only available in *operational* state (see 4.8.3). The corresponding back-up copy of the file in the system directory remains unchanged in that case.

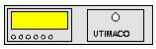
For replacing an existing file in the system directory you have to clear the CryptoServer first (see *BLClear*, chapter 4.7.4) and reload all files using *BLLoadFile*.

Unlike the LoadFile *command (see 4.8.3) offered by the Administration Module the* BLLoadFile *command*

- *does not replace existing files,*
- *stores files two-fold: one copy in the working directory (\FLASH) and one back-up copy in the system directory (\SYS),*
- *should only be used to load the most important firmware modules (SMOS, CMDS, UTIL and ADM), needed to get a minimal working system.*

Syntax	csadm [Dev=<device>] [Password=<password>] InitPrvKey=<keyspec> … BLLoadFile=<file>
Parameters	<device> device specifier (see 4.1.2) <password> if *Initialization Key* is given in encrypted key file: password of encrypted key file: ■ for hidden password entry: string ‘ask’ (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <keyspec> private part of the *Initialization Key* (see 4.1.3*)* <file> any file, usually a firmware module
Example	csadm InitPrvKey=:cs2:cp8:COM1 … … BLLoadFile=c:\firmware\release\exmp.mtc
required state	*initialized* (boot loader mode required)
Authentication	*Initialization Key*
Output	none on success or error message

Note	■ The system directory has only about 400kBytes space to store firmware modules. It is not possible to store the complete set of firmware modules there. ■ If a file cannot be loaded into the system directory in case of memory bottleneck, an error message will be returned. If possible, the file will still be loaded into the working directory.



With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Bootloader functions ➔ Load base firmware modules

4.7.7 ListFiles

This command lists all files stored on the CryptoServer.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.2.

4.7.8 BLSetRTC

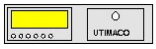
This command sets the CryptoServer's Real Time Clock (RTC).

Instead of the BLSetRTC *command, the* SetTime *command offered by the Administration Module can alternatively be used to set the CryptoServer's clock, see 4.8.6. From the user's point of view there is no difference between these two commands, except that* BLSetRTC *can only be performed in boot loader mode whereas* SetTime *can only be performed in operational mode.*

Syntax	csadm [Dev=<device>] [Password=<password>] … … InitPrvKey | ProdPrvKey=<keyspec> BLSetRTC=<time>
Parameters	<device> device specifier (see 4.1.2) <password> if *Initialization Key* respectively *Production Key* is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <keyspec> InitPrvKey: private part of the *Initialization Key* ProdPrvKey: private part of the *Production Key* *(see 4.1.3 for possible key specifiers)* <time> 'YYYYMMDDHHMMSS' or 'SYSTEM' (see below)
Examples	■ csadm InitPrvKey=c:\keys\init_prv.key BLSetRTC=20011206115530 ■ csadm ProdPrvKey=:cs2:cp8:COM1 BLSetRTC=SYSTEM
required state	*initialized or produced* (boot loader mode required)
Authentication	*Initialization Key or Production Key*
Output	none on success or error message

Note	■ The time format is BCD coded: YYYYMMDDHHMMSS ■ If SYSTEM is given as argument the system time of the host PC is used. ■ The manually entered time string has to be given in the same time zone than the host PC's system time. ■ The CryptoServer's RTC is assumed to run in Greenwich Mean Time (GMT). Whenever *BLSetRTC* is performed, the necessary transformation from the host PC's system time zone to GMT is done automatically by CSADM. ■ Any changing of the clock is taken down on the file 'time.log'. The new entry contains the old time as well as the new time value. The *GetTimeLog* command (see chapter 4.7.14) can be used in any mode to view this file.

The command is not sent to the CryptoServer until authentication is done. If this needs a lot of time (e. g. insertion of a smart card and PIN input), there will be a gap between the given time and the actual time. If CryptoServer's time has to be set very precisely, you can enter a future time and perform the last step (e. g. pressing 'OK' after PIN input) when this time is reached.



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Bootloader Functions` ➔ `Set RTC manually`
respectively (if the RTC should be set to the system time of the Host PC)
`CryptoServer administration` ➔ `Bootloader Functions` ➔ `Set RTC to system time`

4.7.9 GetTime

This command returns the CryptoServer's system time.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.5.

4.7.10 BLResetAlarm

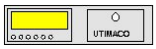
This command manually resets the *alarm state* if the physical reason for the alarm is no longer present:

If an alarm occurs on the CryptoServer, the local Master Key $\mathbf{K_{CS2}}$ and other data will be erased and an entry into the log file 'alarm.log' will be made (see chapter 3.3 *Alarm*). The occurrence of the alarm will be stored as a special alarm state. Even if the physical reason for an alarm has been removed (e. g. a low battery was changed), the alarm state is still active and has to be manually reset by the user:

Except for GetState no other CryptoServer command can be executed until the BLResetAlarm *command has been executed successfully. The* GetState *(see chapter 4.8.1) command shows detailed information about the current alarm.*
If the reason for the alarm is still pending (e. g. foil is still broken), any attempt to reset the alarm state will cause the automatic execution of a reset of the CryptoServer (in the same way the Reset *command does).*

See 7.2 *Alarm Treatment* for more details about what to do in case of an alarm.

Syntax	csadm [Dev=<device>] [Password=<password>] … … InitPrvKey | ProdPrvKey=<keyspec> BLResetAlarm
Parameters	<device> device specifier (see 4.1.2) <password> if *Initialization Key* respectively *Production Key* is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <keyspec> InitPrvKey: private part of the *Initialization Key* ProdPrvKey: private part of the *Production Key* *(see 4.1.3 for possible key specifiers)*
Example	csadm InitPrvKey=c:\keys\init_prv.key BLResetAlarm
required state	*initialized, produced or manufactured* (boot loader mode required)
Authentication	*Initialization Key or Production Key*
Output	none on success or error message
Note	■ If the alarm reason is still present, an error message will be returned and the CryptoServer will execute a reset. ■ Any new alarm is taken down on the 'alarm.log' file, together with the actual time when the alarm happened. The *GetAlarmLog* command (see chapter 4.7.11) can be used to view this file.



With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ *Bootloader Functions* ➔ *Reset Alarm*

4.7.11 GetBootLog

GetBootLog retrieves a log file which contains log messages made by the operating system and other firmware modules during the boot process. The boot log is held in memory and is not written into a file. In this way the content of the previous boot log file is cleared every time the operating system starts.

Log messages can be viewed externally by connecting a PC's serial port with a crossed serial cable to the CryptoServer's serial port 1 (at the bracket of the PCI-card). Use the CSTerm *command of the CSADM tool to view the log messages (see 4.15.4).*

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.9.

4.7.12 GetAlarmLog

The content of the 'alarm.log' file (which is stored in the system directory \SYS) is displayed. Every new alarm is taken down on this log file by the boot loader.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.9.

4.7.13 GetTempLog

The content of the 'temp.log' file (which is stored the system directory \SYS) is displayed.

A new entry is taken down on this log file if CryptoServer's temperature exceeds the range between 5°C and 58°C during the CryptoServer's start-up (i. e. in case of power-on, after the appearance of an alarm or after the execution of *Reset*, *ResetToBL* or *Restart*).

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.10.

4.7.14 GetTimeLog

The content of the 'time.log' file (which is stored in the working directory \FLASH) is displayed. A new entry is taken down on this log file whenever the CryptoServer's clock is set.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.11.

4.7.15 LoadPkg

This command gives the user a facile possibility to load or update a set of several firmware modules and/or files into the CryptoServer in only one step. It thus can replace a series of succeeding CSADM commands.

The *LoadPkg* command loads the contents of the given package file (archive "*.mpkg", see 3.7.2) into the CryptoServer. Depending on the given command line flags the load procedure can also perform a *BLClear* if needed.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.15.

4.7.16 CheckPkg

This command compares the contents of a given package file (archive "*.mpkg", see 3.7.2) against the files actually loaded on a CryptoServer. It announces if the package file contents differ from the loaded firmware and gives information about missing firmware.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.16.

4.7.17 Test

With this command a communication test with the CryptoServer is executed. It sends series of test patterns to the CryptoServer, which echoes the received command. On the host side the received answer is compared with the command originally sent.

This command can be performed in operational mode as well as in boot loader mode. Syntax and usage of the command will therefore be explained in detail in section 4.8.13.

4.8 Commands for Extended Administration

In this chapter the commands offered by the Administration Module (ADM) are described. Unlike the boot loader, the Administration Module allows an extended administration of the CryptoServer.

Except for GetState*,* GetAlarmLog*,* GetTempLog*,* GetTimeLog, LoadPkg *and* CheckPkg *these commands will only be executed if the CryptoServer is in operational mode (see 2.3.4)*

Some of the commands have to be authenticated (e. g. *LoadFile*), some do not (e. g. *ListFiles*).

In *boot loader mode* commands have to be signed either with the *Initialization Key* or the *Production Key* and no distinction is made between different users (only the unnamed system administrator – somebody possessing the required key - is intended to administrate the CryptoServer in boot loader mode).

In *operational mode* command authentication is implemented in a much more detailed way and it is possible to create various users with different permissions, authentication mechanisms and other properties.

For a detail description of the possible authentication mechanisms and the user concept see chapter 2.6. The commands for user management can be found in chapter 4.8.17.

Although command authentication in operational mode is implemented in a very generic way, the way a command has to be authenticated depends on the user, i. e. on his/her special authentication mechanism. Therefore (in this and the following sections) the description of the command syntax for commands in operational mode which have to be authenticated *does not specify each possibility of authentication*. Instead a placeholder (<Authentication>) is inserted, and the command example then shows one possibility of authenticating the command.

If in the syntax of one of the commands described in the following chapters the placeholder <Authentication> *is found,*

- *it has either to be replaced by one or more authentication commands according to the required authentication state and depending on the specific user's (permissions and) authentication mechanism,*
- *or a static login (resulting in the required authentication state) has to be performed before, see 4.10.8.*

*The exact syntax of the various authentication commands (*AuthRSASign, AuthECDSA*,* AuthClrPwd, AuthHMACPwd*,* AuthSha1Pwd*,* AuthRSASC*) can be found in chapter 4.10.*

4.8.1 GetState

This command returns the status of the CryptoServer (see 3.2), its temperature, alarm state, hardware information and set-up information.

The GetState *command is responded by the CryptoServer in any mode (boot loader mode as well as operational mode) and therefore should be executed as first diagnostic measure in case of problems.*
Please prepare the Output of GetState *in case of a support request.*

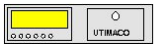
Syntax	csadm [Dev=<device>] GetState
Parameter	<device> device specifier (see 4.1.2)
required state	any
Authentication	none
Output	**CryptoServer is in Operational Mode** **state = 00000005 OPERATIONAL** **temp = 47,5 C** **alarm = OFF** **bl_ver = 01000500** **hw_ver = 01000200** **UID = f4000006 fa1ee701** **adm1 = 5554494d 41434f20 43533030 30303036 | UTIMACO CS000006** **adm2 = 5376656e 27730000 00000000 00000000 | Sven's** **adm3 = 496e6974 2d446576 2d312d4b 65790000 | Init-Dev-1-Key**

The displayed fields have the following meaning:

field	***description***
state	state of the CryptoServer (see also 3.2): ■ Manufactured - on first starting up ■ Produced - after the loading of the *Production Key* ■ Initialized - after the loading of the *Initialization Key* ■ Operational - after the start of the operating system SMOS
temp	temperature of the CryptoServer (see also 3.4):
	< -13°C sensory triggers a temperature alarm => all user data on the CryptoServer will be cleared!

field	*description*	
	< 5°C	during start-up or reset: ■ a new entry is put into the log file ‚temp.log' ■ CryptoServer is set into *power down mode* (see 2.3.4) in running operation: ■ no action
	5°C-58°C	normal temperature range
	> 58°C	during start-up or reset: ■ a new entry is put into the log file ‚temp.log' ■ CryptoServer is set into *power down mode* (see 2.3.4) in running operation: ■ commands will not be executed any longer
	> 66°C	sensory triggers a temperature alarm => all user data on the CryptoServer will be cleared!
alarm	either ON or OFF, if ON (see also 3.3) the following reasons are possible and will be shown in case of an alarm state: ■ Power is too low ■ Power is too high ■ Temperature too high ■ Temperature too low ■ Outer foil is broken ■ Inner foil is broken ■ Invalid Master Key (this usually occurs in case of an empty battery) ■ External Erase is executed (manually by a short-circuit of the corresponding pins on the PCI-card) In addition it is shown if the alarm reason is still present (e. g. foil is still broken) or if it has been removed in the meantime (e. g. empty battery has been replaced).	
bl_ver	version of the boot loader, should be equal or greater than 1.0.5.0	
hw_ver	version of the hardware, should be equal or greater than 1.0.2.0	
UID	8 bytes long, unique ID of the CryptoServer (as a hardware property)	
adm1	16 bytes long, unique serial number of the CryptoServer which is assigned during the production process by Utimaco Safeware AG.	
adm2	16 bytes long field which is assigned by Utimaco Safeware AG with the first loading of the Initialization Key (*BLLoadInitKey* command). Usually the customer's name is registered here.	

field	*description*
adm3	16 bytes long field which can be freely assigned with every change of the *Initialization Key* (*BLChangeInitKey* command, see 4.7.5). This field may be empty or used otherwise, but it is recommended to enter the name of the actual *Initialization Key* here (e. g. 'Init-Live01-Key'). In this way there will never be a question about the actually loaded *Initialization Key*.



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Get status of CryptoServer`

4.8.2 ListFiles

This command lists all files stored on the CryptoServer.

The following directories are available on the CryptoServer:

Component	Directory	Size	Description
flash device (see also 2.1.7.2)	\SYS system directory	496 KBytes	Only the boot loader is able to write onto the system directory, i. e. in operational mode this directory is generally write-protected. The initial administration keys, the boot loader's configuration file, log files and the back-up copies of the firmware modules are stored here.
flash device (see also 2.1.7.2)	\FLASH working directory	15,5 MBytes	Every firmware module has read and write access to the working directory. The regular set of firmware modules and any other kind of application data (databases, configuration or log files, ...) is stored here.
NV-RAM (see 2.1.7.5)	\NVRAM non-volatile directory	512 KBytes	Every firmware module has read and write access to this directory. Often changing data, which still should be stored non-volatile, is stored here.

Don't use the working directory \FLASH to store often changing data (e. g. counter). This will result in a great wear and tear of the flash-component and therefore in a decreased lifetime of the CryptoServer.
Use the \NVRAM directory instead to store this data non-volatile!

Syntax	csadm [Dev=<device>] ListFiles[=<pattern>]
Parameters	<device> device specifier (see 4.1.2) <pattern> filename pattern (search mask, wildcards '*' can be used)
Examples	■ csadm ListFiles ■ csadm ListFiles=exmp.mtc ■ csadm ListFiles=*.mtc ■ csadm ListFiles=SYS\smos.mtc ■ csadm ListFiles=*
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none

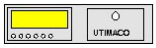
Output

```
SYS\Init.KeyRsaPub      168    -
SYS\MdlSg.KeyRsaPub     168    -
SYS\Prod.KeyRsaPub      168    -
SYS\adm.mtc             55760  ADM    0x087  C64  2.0.0.1  Administration Module
SYS\bl.ini              72     -
SYS\cmds.mtc            78288  CMDS   0x083  C64  1.0.8.1  Command Scheduler SMOS
SYS\smos.mtc            135632 SMOS   0x000  C64  2.0.0.5  Operating System
SYS\util.mtc            61904  UTIL   0x086  C64  2.0.0.2  Utility Module

FLASH\VMBK1.db          113    -
FLASH\adm.mtc           55760  ADM    0x087  C64  2.0.0.1  Administration Module
FLASH\aes.mtc           57804  AES    0x08b  C64  1.0.1.0  AES Module
FLASH\asn1.mtc          37328  ASN1   0x091  C64  1.0.3.1  Asn1 Module
FLASH\cmds.mtc          78288  CMDS   0x083  C64  1.0.8.1  Command Scheduler
FLASH\crgen.mtc         74192  CRGEN  0x0e0  C62  1.4.0.0  C/R Generator for SGE
FLASH\db.mtc            45520  DB     0x088  C64  1.0.1.2  Database module
FLASH\hash.mtc          86480  HASH   0x089  C64  1.0.3.0  Hash Module
FLASH\idea.mtc          61904  IDEA   0x08c  C62  0.9.1.1  IDEA Module
FLASH\lna.mtc           74192  LNA    0x08e  C64  1.0.4.0  Long Number Arithmetic
FLASH\mbk.mtc           66000  MBK    0x096  C64  1.1.0.1  Master Box Key Module
FLASH\pp.mtc            61904  PP     0x082  C64  1.0.5.2  PinPad Driver
FLASH\sc.mtc            41420  SC     0x085  C64  1.0.0.5  Smartcard module
FLASH\sge.db            289    -
FLASH\smos.mtc          135632 SMOS   0x000  C64  2.0.0.5  SMOS Operating System
FLASH\user.db           180    -
FLASH\util.mtc          61904  UTIL   0x086  C64  2.0.0.2  Utility Module
FLASH\vdes.mtc          45520  VDES   0x081  C64  1.0.1.0  DES Module
FLASH\vrsa.mtc          66000  VRSA   0x084  C64  1.0.5.1  RSA Module
```

Note

- Depending on the 'pattern' parameter, information about a dedicated file, a group of files or all files will be returned regarding the CryptoServer*'s* 'FLASH\', 'SYS\' and 'NVRAM\' directory.
- If no pattern is given, all files are listed.
- A wildcard ('*') can be used.
- The CryptoServer's file system is case sensitive!
- If a directory does not contain any file, it will not be displayed.
- 'ListFiles' displays the following information:
  - path\filename
  - file size

  If the file is a firmware module, additionally the following information will be displayed:
  - abbreviation (module's short name)
  - module ID (FC)
  - CPU type
  - version number
  - long name



With the CryptoServer LAN, and if the CryptoServer is in operational mode, this command will be performed by choosing the menu point
`CryptoServer administration ➔ Admin-Module Functions ➔ List Files`

4.8.3 LoadFile

This command loads a file (usually firmware modules) onto the working directory (\FLASH) or the non-volatile directory (\NVRAM) of the CryptoServer.

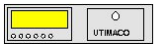
If the given file already exists, it will be replaced. If the loaded file is a firmware module, the existing module will only be replaced if its version is less or equal to the version of the newly loaded module.

The system directory (\SYS) can only be written by the boot loader and thus is write-protected in operational mode. Therefore, if a firmware module in the working directory (\FLASH) is replaced (using this *LoadFile* command), the corresponding back-up copy in the \SYS directory will remain unchanged.

A loaded/replaced firmware module does not become active until the CryptoServer has been restarted (see Restart *command, chapter 4.6.3).*

Syntax	csadm [Dev=<device>] <*Authentication*> LoadFile=<file>[,<dir>]
Parameter	<device> device specifier (see 4.1.2) <file> filename (eventually with path) <dir> CryptoServer directory where the file should be stored (FLASH or NVRAM). If omitted, the FLASH directory is used as default.
Examples	■ csadm AuthRSASign=ADMIN,d:\keys\init_prv.key … … LoadFile=exmp_dbg_1.0.2.3.mtc ■ csadm AuthSHA1Pwd=paul,swordfish LoadFile=usage.cnt,NVRAM
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 6 (see chapters 2.6 and 4.10).
Output	none on success or error message

Note	■ If the filename of a firmware module contains an underline character ('_') the rest of the name up to the file extension will be stripped before loading (e. g. exmp_dbg.mtc -> exmp.mtc, adm_1.0.0.1.mtc -> adm.mtc). ■ If no path is given with the <file> parameter, the file will be taken from the current directory. If the file to be loaded is stored on another directory (e. g. on a floppy), the respective path must be given. ■ The CryptoServer's file system is case sensitive! ■ Filenames of firmware modules (with extension '.mtc') are converted to lower case letter before being loaded onto the CryptoServer. ■ If several files should be loaded in one step, the last part of the command ('LoadFile=...') has to be repeated for every wanted file.



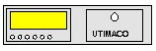
With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ Load File

4.8.4 DeleteFile

With this command a file or a group of files can be deleted from the CryptoServer's working directory (\FLASH) or non-volatile directory (\NVRAM).

The system directory (\SYS) can only be written by the boot loader and is write-protected in operational mode. Thus, if a firmware module in the working directory (\FLASH) has been deleted with DeleteFile*, the corresponding back-up copy in the \SYS directory remains unchanged.*
Even if already deleted from the working directory, a firmware module is still running (in SD-RAM memory)! Only when the CryptoServer is restarted the module becomes inactive (see Restart *command, chapter 4.6.3).*

Syntax	csadm [Dev=<device>] *<Authentication>* DeleteFile=[<dir>\]<file>
Parameters	<device> device specifier (see 4.1.2) <file> filename <dir> CryptoServer directory (FLASH or NVRAM). If omitted, the working directory (\FLASH) is used as default.
Examples	■ csadm AuthSHA1Pwd=paul,swordfish DeleteFile=exmp.mtc ■ csadm AuthRSASign=sm,:usa:cm8:COM2 ... ... DeleteFile=NVRAM\usage.cnt
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 6 (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	■ Wildcards ('*') can be used. ■ The CryptoServer file system is case sensitive! ■ If several files should be deleted in one step, the last part of the command ('DeleteFile=...') has to be repeated for every wanted file, or wildcards have to be used.

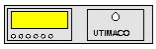


With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Delete File(s)`

4.8.5 GetTime

This command returns the CryptoServer's system time.

Syntax	csadm [Dev=<device>] GetTime
Parameter	<device> device specifier (see 4.1.2)
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**Date: 06.11.2002 Time: 15:35:49.400**
Note	■ The system clock of the CryptoServer has a resolution of 1/1000 second (one millisecond). ■ Date and time are given in the time zone of the host PC's system time. ■ The necessary transformation from the CryptoServer's internal system time (which runs in Greenwich Mean Time, GMT) to the host PC's system time zone is done automatically by CSADM.



With the CryptoServer LAN, and if the CryptoServer is in operational mode, this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ Get time

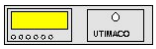
4.8.6 SetTime

This command sets the CryptoServer's system clock.

Instead of this SetTime *command the* BLSetRTC *command, which is offered by the boot loader, can alternatively be used to set the CryptoServer's clock, see 4.7.8. From the user's point of view there is no difference between these two commands, except that* BLSetRTC *can only be performed in boot loader mode whereas* SetTime *can only be performed in operational mode.*

Syntax	csadm [Dev=<device>] *<Authentication>* SetTime=<time>
Parameters	<device> device specifier (see 4.1.2) <time> 'YYYYMMDDHHMMSS' or 'SYSTEM' (see below)
Examples	■ csadm AuthRSASign=ADMIN,:cm8:usa:COM1 SetTime=SYSTEM ■ csadm AuthSHA1Pwd=paul,swordfish SetTime=20020602115500
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 6 (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	■ The time has to be given as digit string: YYYYMMDDHHMMSS where YYYY=year, MM=month, DD=day, HH=hour, MM=minute, SS=second ■ If SYSTEM is given as argument the system time of the host PC is used. ■ The manually entered time string has to be given in the same time zone than the host PC's system time. ■ The CryptoServer's system clock is assumed to run in Greenwich Mean Time (GMT). Whenever *SetTime* is performed, the necessary transformation from the host PC's system time zone to GMT is done automatically by CSADM. ■ Any changing of the clock is taken down on the file 'time.log'. The new entry contains the old time as well as the new time value. The *GetTimeLog* command (see chapter 4.8.11) can be used in any mode to view this file.

The command is not sent to the CryptoServer until authentication is done. If this requires a lot of time (e. g. insertion of a smart card and PIN input) there is a gap between the given time and the actual time. If the CryptoServer's time has to be set very precisely you can enter a future time and perform the last step (e. g. pressing 'OK' after PIN input) when this time is reached



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Set time manually`
respectively (if the RTC should be set to the system time of the Host PC)
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Set time to system time`

4.8.7 ListModulesActive

This function returns a list with information about all firmware modules which are currently loaded by the operating system SMOS, giving their module-ID, abbreviated name, version number and their respective initialization level.

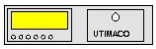
If a module cannot be started or fully initialized, the GetBootLog *command (see next chapter 4.8.8) provides more detailed information about the reason.*

Syntax	csadm [Dev=<device>] ListModulesActive
Parameters	<device> device specifier (see 4.1.2)
required state	*operational*
Authentication	none
Output	**0 SMOS C64 2.0.0.5 INIT_OK** **81 VDES C64 1.0.1.0 INIT_OK** **82 PP C64 1.0.5.2 INIT_OK** **83 CMDS C64 1.0.8.1 INIT_OK** **84 VRSA C64 1.0.5.1 INIT_OK** **85 SC C64 1.0.0.5 INIT_OK** **86 UTIL C64 2.0.0.2 INIT_OK** **87 ADM C64 2.0.0.1 INIT_OK** **88 DB C64 1.0.1.2 INIT_OK** **89 HASH C64 1.0.3.0 INIT_OK** **8b AES C64 1.0.1.0 INIT_OK** **8c IDEA C62 0.9.1.1 INIT_OK** **8e LNA C64 1.0.4.0 INIT_OK** **91 ASN1 C64 1.0.3.1 INIT_OK** **96 MBK C64 1.1.0.1 INIT_OK** **e0 CRGEN C62 1.4.0.0 INIT_OK**
Note	■ Possible module initialization levels are 0 MDL_INIT_NONE 1 MDL_INIT_INTERNAL 2 MDL_INIT_DEP_OK 3 MDL_INIT_OK 4 MDL_INIT_FAILED (see below for the meaning of this levels) ■ If for a module no entry is found, the module is not loaded. ■ After loading or replacing a firmware module, the CryptoServer has to be restarted (see *Restart* command chapter 4.6.3) before the firmware module becomes active.

During the boot process of the CryptoServer, the operating system SMOS starts, after its own initialization, all other firmware modules. The module start is a complex process: Every module that is found and started by SMOS inside the flash file will run through the above given states of initialization, in case of success ending with the MDL_INIT_OK state.

The meaning of the initialization levels is the following:

Initialization level	Description
MDL_INIT_NONE	The module is present but the initialization of the module has not been started yet. This entry will be made by the OS when loading the module into the SD-RAM.
MDL_INIT_INTERNAL	The module has finished its internal initialization (first step of initialization). “Internal” means all initialization tasks that can be done without using services from other modules like memory allocation, FIFO creation, global data initialization, etc.
MDL_INIT_DEP_OK	The module has successfully completed the check of dependencies on other modules (second step of initialization). “Dependencies on other modules” means that the module is dependent on services provided by other modules in order to run correctly. Example: the SC (Smart Card) module requires the PP (PIN-Pad) module to do its work.
MDL_INIT_OK	This is the highest possible level: The initialization of the module is completed (third step of initialization). Possible calls to services from other modules are done successfully.
MDL_INIT_FAILED	**Module initialization failed**. Services provided by this module are not available.



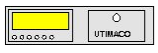
With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ List currently active modules

4.8.8 GetBootLog

GetBootLog retrieves a log file which contains log messages made by the operating system and other firmware modules during the boot process. The boot log is held in memory and is not written into a file. In this way the content of the previous boot log file is cleared every time the operating system starts. The size of the boot log is limited, that is why it contains only log messages made during the boot phase. See also 2.3.3.4.

Log messages can be viewed externally by connecting a PC's serial port with a crossed serial cable to the CryptoServer's serial port 1 (at the bracket of the PCI-card). Use the CSTerm *command of the CSADM tool to view the log messages (see 4.15.4).*

Syntax	csadm [Dev=<device>] GetBootLog
Parameter	<device> device specifier (see 4.1.2)
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**SMOS Ver. 1.0.0.3 started** **module 0x83 (CMDS) initialized successfully** **module 0x89 (HASH) initialized successfully** **module 0x82 (PP) initialized successfully** **module 0x86 (UTIL) initialized successfully** **module 0x8e (LNA) initialized successfully** **module 0x81 (VDES) initialized successfully** **module 0x87 (ADM) initialized successfully** **module 0x84 (VRSA) initialized successfully** **module 0x98 (EMV) initialized successfully** **module 0x85 (SC) initialized successfully** **module 0x91 (ASN1) initialized successfully** **module MBK can't find module DB (00000000)** **FATAL: module 0x96 (MBK) initialization failed (err = b096fe01)**



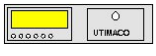
With the CryptoServer LAN, and if the CryptoServer is in operational mode, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Show boot log`

4.8.9 GetAlarmLog

The content of the 'alarm.log' file (which is stored in the system directory \SYS) is displayed.

Every new alarm is taken down on this log file by the boot loader. The 'alarm.log' file is deleted when the CryptoServer is cleared to the *produced* state (see *BLClear*, chapter 4.7.4). If no alarm has occurred since CryptoServer's production, the 'alarm.log' file does not exist.

Syntax	csadm [Dev=<device>] GetAlarmLog
Parameter	<device> device specifier (see 4.1.2)
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**No. Date Time Sens** **----------** **0 22.03.2002 19:06:35 0x027f : ext_Erase** **1 22.03.2002 19:07:12 0x02f7 : Out_foil** If no file 'alarm.log' exists an error message is returned.
Note	The following (combination of) alarms are possible: ■ Pow_low Power is too low ■ Pow_high Power is too high ■ Temp_high Temperature too high ■ Temp_low Temperature too low ■ Out_foil Outer foil is broken ■ In_foil Inner foil is broken ■ ext_Erase External Erase was executed (manually) ■ inval_MK Invalid (corrupted) CryptoServer Master Key $\mathbf{K_{CS2}}$ (this usually occurs in case of an empty battery) See chapter 7.2 '*Alarm Treatment*' how to deal with an alarm.



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration ➔ Admin-Module Functions ➔ View alarm log`
or (if the CryptoServer is in bootloader mode)
`CryptoServer administration ➔ Bootloader Functions ➔ View alarm log`

4.8.10 GetTempLog

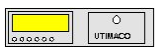
The content of the 'temp.log' file (which is stored in the system directory \SYS) is displayed.

A new entry is taken down on this log file if the CryptoServer's temperature exceeds the range between 5°C and 58°C during the CryptoServer's start-up (i. e. in case of power-on, after the occurrence of an alarm or after the execution of *Reset*, *ResetToBL* or *Restart*). The 'temp.log' file is deleted when the CryptoServer is cleared to the *initialized* state (see *BLClear*, chapter 4.7.4). If the temperature has never been out of range since CryptoServer's last initialization, the file 'temp.log' does not exist.

If the CryptoServer's temperature is lower than 5°C or higher than 58°C the CryptoServer will no longer respond to any command. If it is restarted in this temperature state it will be put into power-down mode and then, after cooling down, must be restarted in order to return to the normal mode.
Exceeding this temperature range does not inevitably lead to an alarm (and CryptoServer's clearing as a result). A temperature alarm does only occur in case of exceeding the range between –13°C and 66°C.

See chapter 3.4 for more details about the CryptoServer's temperature treatment.

Syntax	csadm [Dev=<device>] GetTempLog
Parameter	<device> device specifier (see 4.1.2)
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**No. Date Time Temp Low High** **----------** **0 22.03.2002 19:06:35 58,5 5,0 58,0** **1 22.03.2002 19:07:12 59,0 5,0 58,0** Within one line, the **Temp** temperature gives the measured temperature, whereas **Low** and **High** give the limits of the normal temperature range. If the file 'temp.log' does not exist, an error message will be returned.



With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ View temp log
or (if the CryptoServer is in bootloader mode)
CryptoServer administration ➔ Bootloader Functions ➔ View temp log

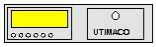
4.8.11 GetTimeLog

The content of the file 'time.log' (which is stored in the working directory \FLASH) is displayed.

A new entry is taken down on this log file any time the CryptoServer's clock is set. The 'time.log' file is not write-protected. It will be deleted any time the CryptoServer is completely cleared (see *BLClear*, chapter 4.7.4) and can also be explicitly deleted by the user (see *DeleteFile* chapter 4.8.4). If the clock was not set since CryptoServer's last initialization, the 'time.log' file does not exist.

Entries are added to this log file by the boot loader (see BLSetRTC*, chapter 4.7.8) as well as by the Administration Module in operational mode (see* SetTime *chapter 4.8.6)*

Syntax	csadm [Dev=<device>] GetTimeLog
Parameter	<device> device specifier (see 4.1.2)
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**No. date time new date new time** **----------** **0 22.10.2002 18:24:57 22.10.2002 18:24:31** **1 25.10.2002 09:45:13 25.10.2002 09:44:59** **2 05.11.2002 13:14:27 05.11.2002 13:14:01** If the 'time.log' file does not exist an error message is returned.



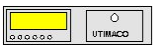
With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ View time log
or (if the CryptoServer is in bootloader mode)
CryptoServer administration ➔ Bootloader Functions ➔ View time log

4.8.12 MemInfo

This function returns information about the current memory usage of the CryptoServer.

The desired directory ('SYS\', 'FLASH\' or 'NVRAM\') may be passed as command parameter. If none of the above directories is given, memory information about all directories will be returned.

Syntax	csadm [Dev=<device>] MemInfo[=<dir>]
Parameters	<device> device specifier (see 4.1.2) <dir> directory on the CryptoServer ('SYS', 'FLASH' or 'NVRAM')
Example	csadm MemInfo=SYS
required state	*operational*
Authentication	none
Output	**SYS** **max_size = 425984** **used_size = 225280** **free_size = 199680** **available_size = 200704**
Note	■ max_size: maximum size of the directory ■ used_size: currently used size ■ free_size: currently free size regarding only already formatted blocks. This value is returned only for diagnostic purposes. It does only show a part of the space available for the user. ■ available_size: size available for the user regarding already free blocks as well as unused space which could be freed if needed



With the CryptoServer LAN this command will be performed by choosing the menu point
CryptoServer administration ➔ Admin-Module Functions ➔ Show memory information

4.8.13 Test

With this command a communication test with the CryptoServer is executed. It sends series of test patterns to the CryptoServer, which echoes the received command. On the host side the received answer is compared with the command originally sent.

Syntax	csadm [Dev=<device>] Test=[<datalength>,]<loopcount>
Parameter	<device> device specifier (see 4.1.2) <datalength> length [bytes] of the used pattern. If this parameter is omitted, 2048 bytes will be used as default value. <loopcount> number of execution cycles.
Examples	■ csadm Test=16,10000 ■ csadm Test=1000000
required state	*initialized* or *operational* (command can be performed in both boot loader mode and operational mode)
Authentication	none
Output	**- Random Block Test** **data length: 16** **loop count : 10000** **10000** **execution completed in 832 ms** **data throughput: 192307 bytes/s** **transaction time: 0.083200 ms**
Note	■ If no data length is given, a 'walking zero' test is performed with a block length of 2048 bytes. ■ If a data length is given, a random data pattern will be generated by the host PC to perform this test. ■ The maximum data length is 256 kBytes ■ A little time is needed to create the test patterns for each loop. A pure benchmark test leads to slightly better results ;-)

4.8.14 ListPinPadApps

Some applications (firmware modules) require a PIN-Pad (smart card reader with display and keyboard) directly connected to CryptoServer's serial port (e. g. for a secure input of a master box key). To easily execute such a command on a CryptoServer LAN, the corresponding function of the firmware module can be registered as a 'PinPad-Application'. Now the command is listed in the 'PinPad applications' menu on CryptoServer LAN's display and can be selected from there (see below).

ListPinPadApps shows all commands registered in this way.

PinPad-Applications are intended to be used on a CryptoServer LAN without the need of connecting a monitor and keyboard to it. The specific PIN-Pad application can be selected via the CryptoServer LAN display (as sub menu point under the `PinPad applications` *menu point, see below), the further user guidance will then be done over the PIN-Pad display.*
Alternatively PinPad-Applications can be executed like any other command using the function code (module ID) and subfunction code (command ID) returned by ListPinPadApps*. The same as above, watch the PIN-Pad display for the further user guidance.*

Syntax	csadm [Dev=<device>] ListPinPadApps
Parameter	<device> device specifier (see 4.1.2)
Output	**1. Create + store MBK on SC** **FC=0xc4 SFC=17 Data=** **2. Import MBK from SC** **FC=0xc4 SFC=18 Data=** **3. PIN change for MBK smartcard** **FC=0xc4 SFC=22 Data=** **4. Copy MBK smartcard** **FC=0xc4 SFC=23 Data=**
Note	■ The registration of a PIN-Pad application has to be implemented during development of the firmware module! ■ A PIN-Pad application can be started with the CSADM tool by entering the command csadm Cmd=FC,SFC,Data where FC, SFC and Data are the parameters from the list of PIN-Pad applications (see also 4.15.2).

With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `PinPad applications`

4.8.15 LoadPkg

This command gives the user a facile possibility to load or update a set of several firmware modules and/or files into the CryptoServer in only one step. It can replace a series of succeeding CSADM commands (see example below).

The *LoadPkg* command loads the contents of the given package file (archive "*.mpkg", see also 3.7.2) into the CryptoServer. Depending on the given command line flags the load procedure can also perform a *BLClear* if needed.

This command can be performed in operational mode as well as in boot loader mode (see 4.7.15).

Syntax	csadm [Dev=<device>] [Password=<password>] InitPrvKey=<InitPrvKey> LoadPkg=<package>[,<flags>[,<rootpassword>]]	
Parameters	<device>	device specifier (see 4.1.2)
	<password>	if *Initialization Key* is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)*
	<InitPrvKey>	key specifier for private part of the *Initialization Key* (see 4.1.3*)*
	<package>	input package file containing CryptoServer firmware modules and files (filename must have extension ".mpkg")
	<flags>	See below. Flags can be added (+) if this makes sense.
	<rootpassword>	root password of CryptoServer LAN (see note below): ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password This parameter is only relevant if the CryptoServer LAN is used, and if the driver commands *Reset*, *Restart* and *ResetToBL* are password protected (see below).
Example	csadm InitPrvKey=:cs2:cp8:COM1 LoadPkg=firmware.mpkg,NoClear+NoDelete	
required state	*operational* or *initialized*	
Authentication	This command has to be authenticated with the *Initialization Key* (as given in the command syntax).	
Output	information about all actually performed steps and changes in status, mode or firmware module versions (see examples below)	

Note	In case that the CryptoServer LAN is used and the driver commands *Reset*, *Restart* and *ResetToBL* are protected by a root password (i. e. depending on the settings in the LAN box configuration file), the 'password' parameter is mandatory. See section 4.6.

This command will check the MMC and MTC signatures of all firmware modules contained in the package file. As default, for all firmware modules with correct MMC signature but missing or not-verifiable MTC signature the MTC will be re-signed with the given Initialization Key before being loaded into the CryptoServer.
If this default setting should be changed one of the appropriate flags (`NoReSign` or `ForceReSign`, see below) has to be chosen.

If the 'NoDelete' flag is not explicitly set, modules that are stored inside the CryptoServer but not contained in the given package file will be deleted.

flag	**meaning**
`NoClear`	No *BLClear* will be performed at any time (default).
`AllowClear`	A *BLClear* will be performed if necessary.
`ForceClear`	A *BLClear* will be performed before the files are loaded, i. e. all firmware and data inside the CryptoServer will be deleted before the contents of the package file are loaded.
`NoDelete`	Modules not contained in the given archive won't be deleted on the CryptoServer (for default see above).
`NoReSign`	The firmware modules contained in the archive will not be re-signed with the given Initialization Key (for default see above).
`ForceReSign`	*All* firmware modules contained in the archive are automatically re-signed with the given Initialization Key.
`SwapComCreate`	Creates the file FLASH\swap.com in the CryptoServer
`SwapComDelete`	Deletes the file FLASH\swap.com in the CryptoServer (if available)
`SwapComDetect`	If the addressed CryptoServer is a CSLAN, the file FLASH\swap.com will be deleted. For all other CryptoServer this file will be created.

A firmware module contained in the package file will also be loaded (and thus replace an existing module of the same name) if its version number is lower than the version of the module currently loaded on the CryptoServer, i. e. in this case a firmware downgrade would be performed.
However, the module will not be replaced if the version number is the same.

Example 1:

The firmware which is currently stored in the CryptoServer shall be completely replaced by the firmware (and other files) contained in a CryptoServer package file "firmware.mpkg". In this case a *BLClear* should be performed as part of the *LoadPkg* command, i. e. the 'ForceClear'-flag has to be set. During command execution, information about all actually performed steps and the CryptoServer's state and mode will be displayed:

```
   csadm InitPrvKey=:cs2:cp8:COM1 LoadPkg=firmware.mpkg,ForceClear
I: CryptoServer remains in operational mode (state OPERATIONAL)
I: Reset CryptoServer to bootloader mode
I: Delete all files/modules inside the CryptoServer (perform BLClear)
I: Load base firmware modules (ADM, CMDS, SMOS, UTIL)...
I: Starting CryptoServer operating system (perform StartOS)
I: CryptoServer remains in operational mode (state OPERATIONAL)
I: Load file aes_1.0.1.0.mtc
I: Load file asn1_1.0.1.2.mtc
I: Load file db_1.0.1.1.mtc
I: Load file hash_1.0.1.0.mtc
I: Load file lna_1.0.3.0.mtc
I: Load file pp_1.0.5.0.mtc
I: Load file sc_1.0.0.5.mtc
I: Load file vdes_1.0.0.3.mtc
I: Load file vrsa_1.0.4.0.mtc
I: Restarting CryptoServer
I: CryptoServer remains in operational mode (state OPERATIONAL)
I: The CryptoServer was successfully activated
Package fw1.mpkg successfully loaded
```

Example 2:

In the following example some firmware modules will be upgraded (AES, CMDS), others will be downgraded (VRSA), some modules (that have not been yet available on the CryptoServer) will be loaded, others (that are not contained in the package file) will be deleted from the CryptoServer.

```
   csadm InitPrvKey=:cs2:cp8:COM1 LoadPkg=fw1.mpkg
 I: CryptoServer remains in operational mode (state OPERATIONAL)
 I: Version of firmwaremodule AES in FLASH directory differs with
    configuration (1.0.0.0 vs. 1.0.1.0)
 I: Load file aes_1.0.1.0.mtc
 I: Version of firmwaremodule CMDS in FLASH directory differs with
    configuration (1.0.5.1 vs. 1.0.6.0)
 I: Load file cmds_1.0.6.0.mtc
 I: Version of firmwaremodule VRSA in FLASH directory differs with
    configuration (1.0.4.1 vs. 1.0.4.0)
 I: Delete file FLASH\vrsa.mtc
 I: Load file vrsa_1.0.4.0.mtc
 I: Load file asn1_1.0.1.2.mtc
 I: Load file hash_1.0.1.0.mtc
 I: Delete file FLASH\example.mtc
 I: Restarting CryptoServer
 I: CryptoServer remains in operational mode (state OPERATIONAL)
 I: The CryptoServer was successfully activated
 Package fw1.mpkg successfully loaded
```

4.8.16 CheckPkg

This command compares the firmware modules contained in a given package file (archive "*.mpkg", see 3.7.2) against the firmware currently loaded on a CryptoServer. It announces if the package file contents differ from the loaded firmware and gives detailed information about which modules had to be loaded or deleted (in case the package file is loaded) and about differences in firmware module versions. If a *BLClear* would be unavoidable in case the given package file is loaded, this would be announced, too.

Information about differences concerning files other than firmware is not given.

This command can be performed in operational mode as well as in boot loader mode (see 4.7.16).

Syntax	csadm [Dev=<device>] CheckPkg=<package>[,<password>]	
Parameters	<device>	device specifier (see 4.1.2)
	<package>	input package file containing CryptoServer firmware modules and files (filename must have extension ".mpkg")
	<password>	root password of CryptoServer LAN (see below): ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password This parameter is only relevant if the CryptoServer LAN is used, and if the driver commands *Reset*, *Restart* and *ResetToBL* are password protected (see below).
Example	csadm CheckPkg=firmware.mpkg	
required state	*operational* or *initialized*	
Authentication	none	
Output	information about differences between loaded firmware and firmware contained in given package file (see example below)	
Note	In case that a CryptoServer LAN is used and the driver commands *Reset*, *Restart* and *ResetToBL* are protected by a root password (i. e. depending on the settings in the LAN box configuration file), the 'password' parameter is mandatory. See section 4.6.	

Example:

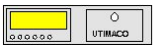
```
   csadm CheckPkg=fw2.mpkg

I: CryptoServer remains in operational mode (state OPERATIONAL)
I: Version of firmwaremodule CMDS in SYS directory differs with
   configuration (1.0.6.0 vs. 1.0.5.1)
I: To update this firmwaremodule a BLClear must be performed
I: Version of firmwaremodule AES in FLASH directory differs with
   configuration (1.0.1.0 vs. 1.0.0.0)
E: The file FLASH\asn1.mtc must be deleted
I: Version of firmwaremodule CMDS in FLASH directory differs with
```

```
   configuration (1.0.6.0 vs. 1.0.5.1)
E: The file FLASH\hash.mtc must be deleted
E: The firmwaremodule PP must be loaded
E: The firmwaremodule SC must be loaded
I: Version of firmwaremodule VRSA in FLASH directory differs with
   configuration (1.0.4.0 vs. 1.0.4.1)
I: Base firmware differs from archive content.
Verifying package fw2.mpkg failed
Error B9068014
      CryptoServer admin library
      PKG Command Interface
      CryptoServer firmware module(s) differs from archive content
```

4.8.17 LoadFWDecKey

This command loads a private RSA key into the CryptoServer that is used to decrypt encrypted firmware modules.

Syntax	csadm [Dev=<device>] <*Authentication*> LoadFWDecKey=<keyspec>
Parameter	<device> device specifier (see 4.1.2) <keyspec> key specifier (see 4.1.3) of firmware decrsyption key may be either a pathname to a file containing the private key or a smartcard descriptor if the key is stored as two XOR-halves on back-up smartcards.
Examples	■ csadm AuthRSASign=ADMIN,d:\keys\init_prv.key … … LoadFWDecKey=C:\keys\fw_dec.key ■ csadm AuthSHA1Pwd=paul,swordfish … … LoadFWDecKey=:cs2:cp8:COM1
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 6 (see chapters 2.6 and 4.10).
Output	none on success or error message



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `Load FW Dec Key`

4.9 User Management

The user management described in this chapter is only available in *operational mode* where command authentication is implemented in a much more extensive way than in *boot loader mode*, which knows only one unnamed user (the system administrator) and two initial keys for command authentication (*Production Key, Initialization Key*).

In operational mode security relevant commands can be authenticated by different users (see chapters 2.5 and 2.6 for the details). In dependency on the command the user therefore has to be fitted out with certain permissions. Some commands require the authentication by two users (2-persons-rule).

New users can be set up on the CryptoServer by adding them to the user database 'user.db', which is hosted on the CryptoServer and managed by the Command Scheduler firmware module CMDS (cmds.mtc). The respective commands are described in this chapter. Within this user management, the following properties can be assigned to each user:

Property	***Description***
Name	user name, up to 255 printable characters (must not contain a '~')
Permission	A user's permission consists of 8 different figures (enumerated downward 7 .. 0) with values between 0 and 3 where each figure stands for a specific user group (which corresponds to a specific group of commands), and the specific value (0, 1, 2 or 3) indicates the level of authentication the user possesses within this user group: '0' means 'no permission' whereas '3' is the highest possible level of authentication. See 2.6 for more details. **Example:** A user with permission '11200000' has the authentication level '1' in the user groups 7 and 6, level '2' in user group 5 and no permission in all other user groups (4 ... 0). User groups 6 and 7 are already predefined: User Group 7: commands for user management and session management, see this chapter and chapter 4.11 User Group 6: commands for extended administration, see 4.8 All other user groups (5 ... 0) can be used for customer's applications.

Mechanism	■ RSA signature ■ ECDSA signature ■ clear password ■ SHA-1 hashed password ■ HMAC password ■ local smart card (i. e. local confirmation of command execution, using a RSA smart card over a smart card reader which is connected directly to the CryptoServer) See 2.6 for more details about the mechanisms and their advantages / disadvantages.
Flags	■ no_login: user has to authenticate each (sensitive) command separately ■ allow_login: user is allowed to login statically (and execute subsequent commands without renewed authentication, see 4.10.8) ■ sm: user may open a secure messaging session without authentication (see 4.11). ■ sma: user may open a secure messaging session if he/she authenticates the command (see 4.11).

Attributes	A string containing a list of assignments with syntax:
	`<AttrName>=<AttrValue>,<AttrName>=<AttrValue>,…`
	Example:
	`P11Slot=005,P11User=SO`
	The meaning of the Attributes depends on the application loaded into the CryptoServer.

HashAlgo	Hash algorithm to be used for RSA, ECDSA or HMAC authentication (in case that not the default hash algorithm shall be used for this user, which is SHA-1 for RSA signatures and SHA-256 for ECDSA signatures and HMAC). Value can be one of the following algorithms: ■ SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512, ■ MD5, ■ RIPEMD-160

Since the creation of new users also requires an authentication (by one or two users with the permission to manage users: authentication level 2 in user group 7) one initial user ‘ADMIN’ is always available and uses the ‘RSA-Signature’ authentication mechanism with the *Initialization Key*. ADMIN is therefore intended to be the CryptoServer’s System Administrator.

ADMIN is always present (even if the database ‘user.db’ has not yet been created) and cannot be deleted. ADMIN has the permission of extended administration (Administration module ADM) and user management (Command Scheduler CMDS) with level 2, i. e. the

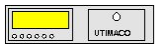
permission is ‘22000000’. This means that he is able to execute commands which otherwise would require two users authenticating the command (2-Persons-Rule).

In operational mode the initial user ‘ADMIN’ is always present and has the permission for ‘User Management’ and ‘Extended Administration’. He authenticates commands with the ‘RSA-Signature’-mechanism using the Initialization Key. Therefore the System Administrator as the owner of the Initialization Key is intended to be the user ‘ADMIN’.

4.9.1 ListUser

This command lists all existing users from the 'user.db' database.

Syntax	csadm [Dev=<device>] ListUser				
Parameter	<device> device specifier (see 4.1.2)				
required state	*operational*				
Authentication	none				
Output	**Name**	**Permission**	**Mechanism**	**Flags**	**Attributes**
	ADMIN	**22000000**	**RSA sign**	**no_login + sma**	**H[SHA-256]**
	sm	**00000022**	**sha-1 passwd**	**allow_login + sm**	
	paul	**22000022**	**sha-1 passwd**	**allow_login**	
	test	**21000011**	**sha-1 passwd**	**allow_login**	**A[P11Slot=007]**
Note	■ The initial user 'ADMIN' will always be displayed (even if the database 'user.db' is not present) and cannot be deleted. ■ For a description of the properties (*name, permission, mechanism, flags, attributes*) see the previous pages or 2.5. *Attributes* may be application-depending attributes, marked by a tag 'A', or non-default hash algorithms to be used for the authentication mechanism of that user, marked by tag 'H'.				



With the CryptoServer LAN this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Admin-Module Functions` ➔ `List Users`

4.9.2 AddUserRSASign

With this command a new user is added to the user database using 'RSA-Signature' as authentication mechanism. Therefore the user needs a RSA key pair either on a smart card or as key file.

During the execution of this command, the public part of the RSA key is read from the smart card or key file and stored in the CryptoServer's user database.

The CryptoServer can check later the authenticity of commands by verifying the RSA command signature: this RSA signature is calculated by the host over a random value (a challenge value retrieved from the CryptoServer in a prior step) and the command data block with the private part of the user's key (PKCS#1 signature format). This signature will then be transmitted to the CryptoServer who will verify it with the help of the RSA key's public part which is stored in the user database.

This mechanism is particularly qualified for communication via Ethernet (CryptoServer LAN) and will therefore be recommended for secure applications.

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserRSASign=<user>,<permission>,<flag1>[+<flag2>],<keyspec>	
Parameters	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6): 8 digits 'XXXXXXXX', each representing a permission level in a specific user group. A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	'sm' / 'sma'
	<keyspec>	public part of the user's RSA key (see 4.1.3) A name of a hash algorithm, put into curly braces { }, can be appended or prepended to the keyspecifier, see table with user properties in 4.8.17 above and example below.
Examples	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 … … AddUserRSASign=Paul,01000000,allow_login+sm,:cs2:cp8:COM1 csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 … … AddUserRSASign=Eve,010{P11Slot=007},sm,{SHA-256}key1.key	
required state	*operational*	
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).	

Output	none on success or error message
Note	If the authentication of this command requires a smart card and the source of the new user's public key is a smart card, too, watch the PIN-Pad's display and 4. insert the smart card containing the new user's RSA key at first, 5. insert the smart card for command authentication after that.

4.9.3 ChangeUserRSASign

With this command a user using the authentication mechanism 'RSA-Signature' changes his RSA Key.

The user needs a new RSA key pair (on smart card or as key file) as well as his/her old RSA Key.

A user is not allowed to change his authentication mechanism, permission or flags.

Syntax	csadm [Dev=<device>] *<Authentication>* … ..... ChangeUserRSASign=<user>,<keyspec>
Parameter	<device> device specifier (see 4.1.2) <user> existing user name <keyspec> public part of the user's new RSA key (see 4.1.3)
Example	csadm AuthRSASign=paul,:cs2:cp8:COM1 ChangeUserRSASign=paul,:cs2:cp8:COM1
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism 'RSA Signature' (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	If the authentication of this command requires a smart card and the source of the user's new public key is a smart card too, watch the PIN-Pad's display and... 1. insert the smart card containing the user's new RSA key at first, 2. insert the smart card for command authentication (old key) after that.

4.9.4 AddUserClrPwd

With this command a new user is added to the user database using 'Clear Password' as authentication mechanism.

During the execution of this command, the entered password is stored in the CryptoServer's user database. The CryptoServer can check later the authentication of commands by comparing the password added to the command with the password stored in its database.

This authentication mechanism does not provide a high level of security, because the plain password is added to each command. A sniffer attack reveals the password very easily.
It is highly recommended to use either another authentication mechanism (e. g. the SHA-1 hashed password mechanism) or secure messaging to encrypt the command data.

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserClrPwd=<user>,<permission>,<flag1>[+<flag2>],<password>	
Parameters	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6): 8 digits 'XXXXXXXX', each representing a permission level in a specific user group. A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	sm' / 'sma'
	<password>	for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: password of the new user (length between 6 and 16 characters)
Examples	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserClrPwd=paul,01000000,no_login+sma,ask csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserClrPwd=paul,01000000,no_login,swordfish csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserClrPwd=paul,00020000{P11Slot=007},no_login,enigma	
required state	*operational*	

Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).
Output	none on success or error message

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the new user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.9.5 ChangeUserClrPwd

With this command a user using the authentication mechanism 'Clear Password' changes his password.

A user is not allowed to change his authentication mechanism, permission or flags

Syntax	csadm [Dev=<device>] *<Authentication>* … ... ChangeUserClrPwd=<user>,<password>
Parameters	<device> device specifier (see 4.1.2) <user> existing user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's new password (length between 6 and 16 characters)
Examples	■ csadm AuthClrPwd=paul,swordfish ChangeUserClrPwd=paul,sesame ■ csadm AuthClrPwd=paul,ask ChangeUserClrPwd=paul,ask
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	The user already has to be present in the user database.

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's new password separately (i. e. a request 'Enter New Passphrase:' follows in one of the next command lines) and hide the entrance on the monitor by the display of default characters.

4.9.6 AddUserSHA1Pwd

With this command a new user is added to the user database using the 'SHA-1 hashed Password' authentication mechanism.

While executing this command, the entered password is stored in the CryptoServer's user database.

As long as this AddUserSHA1Pwd *command is not performed with Secure Messaging (see 4.11), the password will be transmitted in clear to the CryptoServer. Therefore the usage of Secure Messaging is strongly recommended!*

The CryptoServer can check later the command authentication via SHA-1 hash value which is going to be added to each command: This hash value has been calculated by the host based on a random value (a challenge value retrieved from the CryptoServer in a prior step), the password and the command data block. The CryptoServer can recalculate and check the hash value with the help of the password stored in its user database.

Compared with the clear password authentication, this mechanism has the following advantages:

- *With a that way authenticated command, the password will not be submitted in clear and thus cannot be scanned.*
- *Because of the random value a 'Playback'-Attack of the command becomes impossible.*

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserSHA1Pwd=<user>,<permission>,<flag1>[+<flag2>],<password>	
Parameters	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6): 8 digits 'XXXXXXXX', each of them representing a permission level in a specific user group. A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	'sm' / 'sma'
	<password>	for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: new user's password (length between 6 and 16 characters)

Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserSHA1Pwd=paul,11000021,allow_login,swordfish csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserSHA1Pwd=paul,11000021,allow_login+sm,ask csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserSHA1Pwd=paul,00000020{P11Slot=004},no_login+sm,ask
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).
Output	none on success or error message

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the new user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.9.7 ChangeUserSHA1Pwd

With this command a user with the authentication mechanism 'SHA1 hashed Password' changes his password.

A user is not allowed to change his authentication mechanism, permission or flags.

As long as this ChangeUserSHA1Pwd *command is not performed with Secure Messaging (see 4.11), the new password will be transmitted in clear to the CryptoServer. Therefore the usage of secure messaging is strongly recommended.*

Syntax	csadm [Dev=<device>] *<Authentication>* … ... ChangeUserSHA1Pwd=<user>,<password>
Parameter	<device> device specifier (see 4.1.2) <user> existing user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's new password (length between 6 and 16 characters)
Example	■ csadm AuthSHA1Pwd=paul,swordfish ChangeUserSHA1Pwd=paul,sesame ■ csadm AuthSHA1Pwd=paul,ask ChangeUserSHA1Pwd=paul,ask
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	The user has to be already present in the user database

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's new password separately (i. e. a request 'Enter New Passphrase:' follows in one of the next command lines) and hide the entrance on the monitor by the display of default characters.

4.9.8 AddUserRSASC

With this command a new user is added to the user database using 'RSA Smart Card' as authentication mechanism. Therefore the user needs a RSA key pair on a smart card, i. e. a RSA signature smart card.

While executing this command, the public part of the RSA key is read from the smart card and stored in the CryptoServer's user database. The CryptoServer can check later the authentication of commands by verifying the signature of a challenge value made with the private part of the user's key. For generating this signature the PIN-Pad (including smart card reader) has to be connected *directly* to CryptoServer's serial port 2 and the user's smart card has to be inserted into the reader.

Therefore this authentication mechanism is only applicable for local applications.

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserRSASC=<user>,<permission>,<flag1>[+<flag2>],<keyspec>	
Parameter	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6):
		8 digits 'XXXXXXXX', each representing a permission level in a specific user group.
		A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	'sm' / 'sma'
	<keyspec>	public part of the user's RSA key (see 4.1.3)
Examples	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserRSASC=paul,01000000,allow_login+sm,:cs2:cp8:COM1 csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserRSASC=paul,0010{P11Slot=004},sm,:cs2:cp8:COM1	
required state	*operational*	
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).	
Output	none on success or error message	

Note	■ The PIN-Pad has to be connected to a serial line of the computer where the CSADM tool is running. ■ If the authentication of this command requires a smart card, watch the PIN-Pad display and ... 1. insert the smart card containing the new user's RSA key at first, 2. insert the smart card for the command authentication after that.

4.9.9 ChangeUserRSASC

With this command a user with the authentication mechanism 'RSA Smart Card' changes his RSA Key.

The user needs a new RSA key pair (RSA signature smart card) as well as his old RSA Key.

A user is not allowed to change his authentication mechanism, permission or flags

Syntax	csadm [Dev=<device>] <*Authentication*> … ... ChangeUserRSASC=<user>,<keyspec>
Parameter	<device> device specifier (see 4.1.2) <user> existing user name <keyspec> public part of the user's new RSA key (see 4.1.3)
Example	csadm AuthRSASC=paul,:cs2:cp8:COM1 ... ... ChangeUserRSASC=paul,:cs2:cp8:COM1
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	■ If the user's new RSA key is read from a smart card then two PIN-Pads are needed: a) one connected to the computer where the CSADM tool is running to insert the smart card containing the user's new RSA key, b) a second PIN-Pad to insert the smart card for the command authentication (with old key) after that. ■ If the new user's RSA key is read from a file, only one PIN-Pad is necessary for authentication with the old key.

4.9.10 AddUserHMACPwd

With this command a new user is added to the user database using the 'HMAC Password' authentication mechanism.

While executing this command, the entered password is stored in the CryptoServer's user database.

As long as this AddUserHMACPwd *command is not performed with Secure Messaging (see 4.11), the password will be transmitted in clear to the CryptoServer. Therefore the usage of Secure Messaging is strongly recommended!*

The CryptoServer can check later the command authentication via HMAC value which is going to be added to each command: This hash value has been calculated by the host based on a random value (a challenge value retrieved from the CryptoServer in a prior step) and the command data block using the password as the HMAC key. The CryptoServer can recalculate and check the hash value with the help of the password stored in its user database.

Compared with the clear password authentication, this mechanism has the following advantages:

- *With a that way authenticated command, the password will not be submitted in clear and thus cannot be scanned.*
- *Because of the random value a 'Playback'-Attack of the command becomes impossible.*

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserHMACPwd=<user>,<permission>,<flag1>[+<flag2>],<password>	
Parameters	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6): 8 digits 'XXXXXXXX', each of them representing a permission level in a specific user group. A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	'sm' / 'sma'

	<password>	for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: new user's password A name of a hash algorithm, put into curly braces { }, can be appended or prepended to the password, see table with user properties in 4.8.17 above and example below.
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserHMACPwd=paul,11000021,allow_login,swordfish csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserHMACPwd=paul,11000021,allow_login+sm,ask csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserHMACPwd=paul,020{P11Slot=004},sm,{SHA-512}enigma	
required state	*operational*	
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).	
Output	none on success or error message	

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the new user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.9.11 ChangeUserHMACPwd

With this command a user with the authentication mechanism 'HMAC Password' changes his password.

A user is not allowed to change his authentication mechanism, permission or flags.

As long as this ChangeUserHAMCPwd *command is not performed with Secure Messaging (see 4.11), the new password will be transmitted in clear to the CryptoServer. Therefore the usage of secure messaging is strongly recommended.*

Syntax	csadm [Dev=<device>] *<Authentication>* … ... ChangeUserHMACPwd=<user>,<password>
Parameter	<device> device specifier (see 4.1.2) <user> existing user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's new password
Examples	csadm AuthHMACPwd=paul,swordfish ChangeUserHMACPwd=paul,sesame csadm AuthHMACPwd=paul,ask ChangeUserHMACPwd=paul,ask
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	The user has to be already present in the user database

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's new password separately (i. e. a request 'Enter New Passphrase:' follows in one of the next command lines) and hide the entrance on the monitor by the display of default characters.

4.9.12 AddUserECDSA

With this command a new user is added to the user database using 'ECDSA-Signature' as authentication mechanism. Therefore the user needs an ECDSA key pair either on a smart card or as key file.

During the execution of this command, the public part of the ECDSA key is read from the smart card or key file and stored in the CryptoServer's user database.

The CryptoServer can check later the authenticity of commands by verifying the ECDSA command signature: this ECDSA signature is calculated by the host over a random value (a challenge value retrieved from the CryptoServer in a prior step) and the command data block with the private part of the user's key. This signature will then be transmitted to the CryptoServer who will verify it with the help of the ECDSA key's public part which is stored in the user database.

This mechanism is particularly qualified for communication via Ethernet (CryptoServer LAN) and will therefore be recommended for secure applications.

Syntax	csadm [Dev=<device>] <*Authentication*> … ... AddUserECDSA=<user>,<permission>,<flag1>[+<flag2>],<keyspec>	
Parameters	<device>	device specifier (see 4.1.2)
	<user>	user name
	<permission>	user's permissions (see 2.6): 8 digits 'XXXXXXXX', each representing a permission level in a specific user group. A list of attributes, put into curly braces { }, can be appended to the permission digits, see table with user properties in 4.8.17 above and example below.
	<flag1>	'no_login' / 'allow_login'
	<flag2>	'sm' / 'sma'
	<keyspec>	public part of the user's ECDSA key (see 4.1.3) A name of a hash algorithm, put into curly braces { }, can be appended or prepended to the keyspecifier, see table with user properties in 4.8.17 above and example below.
Examples	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 … … AddUserECDSA=Paul,01000000,allow_login+sm,:cs2:cp8:COM1 csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 … … AddUserECDSA=Eve,010{P11Slot=007},sm,{SHA-512}key1.key	
required state	*operational*	
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).	

Output	none on success or error message
Note	If the authentication of this command requires a smart card and the source of the new user's public key is a smart card, too, watch the PIN-Pad's display and 1. insert the smart card containing the new user's RSA key at first, 2. insert the smart card for command authentication after that.

4.9.13 ChangeUserECDSA

With this command a user using the authentication mechanism 'ECDSA-Signature' changes his ECDSA Key.

The user needs a new ECDSA key pair (on smart card or as key file) as well as his/her old ECDSA Key.

A user is not allowed to change his authentication mechanism, permission or flags.

Syntax	csadm [Dev=<device>] <*Authentication*> … ..... ChangeUserECDSA=<user>,<keyspec>
Parameter	<device> device specifier (see 4.1.2) <user> existing user name <keyspec> public part of the user's new ECDSA key (see 4.1.3)
Example	csadm AuthECDSA=paul,:cs2:cp8:COM1 ChangeUserECDSA=paul,:cs2:cp8:COM1
required state	*operational*
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism 'ECDSA Signature' (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	If the authentication of this command requires a smart card and the source of the user's new public key is a smart card too, watch the PIN-Pad's display and... 1. insert the smart card containing the user's new ECDSA key at first, 2. insert the smart card for command authentication (old key) after that.

4.9.14 DeleteUser

This function deletes a user from the user database.

Syntax	csadm [Dev=<device>] *<Authentication>* DeleteUser=<user>
Parameter	<device> device specifier (see 4.1.2) <user> existing user
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 DeleteUser=paul
required state	*operational*
Authentication	Command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).
Output	none on success or error message

4.9.15 BackupUser

This command backs up all users that are currently set up on the CryptoServer into a file. Each user entry in the created backup file is encrypted with the CryptoServer's Master Box Key (MBK) and therefore can be handled without additional security measures.

The user 'ADMIN' will not be backed up.

Syntax	csadm [Dev=<device>] <*Authentication*> BackupUser=<file>[,<flags>]
Parameter	<device> device specifier (see 4.1.2) <file> path and file name of the backup file to be created <flags> optional flags: • 'overwrite': backup file will be overwritten if already existing
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 ... ... BackupUser=d:\temp\cs123456-20051212.ubk,overwrite
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	■ If the overwrite flag is not given and the backup file already exists the backup will not be performed and the existing backup file will not be overwritten.

4.9.16 RestoreUser

This command restores the users on the CryptoServer from a backup file (see command *BackupUser*).

The same Master Box Key (MBK) which has been used to create the backup file has to be loaded on the CryptoServer.

Syntax	csadm [Dev=<device>] *<Authentication>* … ... RestoreUser=<file>[,<flags>]
Parameter	<device> device specifier (see 4.1.2) <file> path and file name of the backup file <flags> optional flags: • 'overwrite': if a user with one of the given user names already exists on the CryptoServer it will be overwritten
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 ... ... RestoreUser=d:\temp\cs123456-20051212.ubk,overwrite
required state	*operational*
Authentication	The command must be authenticated with level 2 in user group 7 (see chapters 2.6 and 4.10).
Output	none on success or error message
Note	■ If the overwrite flag is not given and one of the users (identified by its user name) already exists on the CryptoServer the command fails and returns an error code.

4.10 Command Authentication

Whereas the *boot loader mode* knows only one unnamed user (the system administrator) to authenticate commands, the *operational mode* allows a much more extensive command authentication with a variety of authentication mechanisms. Security relevant commands may be authenticated by different users. Depending on the command the user has therefore to be fitted out with certain permissions.

See chapter 2.6 for a detailed explanation of the authentication concept.

Most pre-defined security-relevant commands require the authentication level 2 in a specific user group. Since each user usually has only permission level 1 in one or more user groups (apart from the pre-defined user ADMIN who has permission level 2 in user group 7 and 6, i. e. for the user management and the administrative commands), this means that two users of the specific user group are necessary to authenticate the command. For this reason it can (and will) be said that the specific command requires authentication according to the **2-persons-rule**, i. e. authentication by two independent users is necessary if the command shall be executed.

*If a command requires an authentication according to the **2-persons-rule**, both users have to apply their authentication prior to command execution:*

csadm <*Authentication1*> <*Authentication2*> <command>

The two users may even use different authentication mechanisms.

Example:

csadm AuthRSASign=roberta,:cs2:cp8:COM1 AuthSHA1Pwd=paul,swordfish DeleteFile=..

Another way to use authentication is a *static login* (*Login* command, see chapter 4.10.8). This means that the CryptoServer stores the authentication (by adding the user's permissions to its present authentication state until an explicit *Logoff* command is performed, see 4.10.9), so that all commands (which require this resulting authentication state) issued between the *Login* and the *Logoff* do not require an explicit authentication.

***In no case the Login command should be used if it cannot be made sure that** - during the time when a user is statically logged in - **no unauthorized user is able to send commands to the CryptoServer.** (This is e. g. important when the CryptoServer is connected to a LAN.)*

For security reasons the main administrator ADMIN is not allowed to use static login.

In the following chapters the syntax of all authentication mechanisms is explained. These 'authentication commands' (leading to the necessary authentication state) can be inserted for the placeholder <*Authentication*> which are given in the syntax of all security relevant commands described in this chapter 4.

4.10.1 AuthRSASign

With this command a user with the authentication mechanism 'RSA-Signature' authenticates a single command.

The following steps are performed during command execution:

3. A challenge value is requested from the CryptoServer.
4. Command data and challenge value are signed (according to PKCS#1) using the private part of the user's RSA key.
5. Command data and signature are sent to the CryptoServer.
6. The CryptoServer verifies the signature using the public part of the user's RSA key from its user database.
7. If the verification has been successful (and if the necessary authentication state has been achieved), the CryptoServer will execute the command

Syntax	csadm [Dev=<device>] [Password=<password>] … … AuthRSASign=<user>,<keyspec> [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <password> if user's RSA key is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <user> user name <keyspec> private part of the user's RSA key (smart card or key file, see 4.1.3) In case the user uses a non-default hash algorithm for his RSA Signature authentication mechanism, the name of this hash algorithm, put into curly braces { }, has to be appended or prepended to the keyspecifier (see table with user properties in 4.8.17 above and example below). <command> command which has to be authenticated
Examples	csadm AuthRSASign=paul,:cs2:cp8:COM1 DeleteFile=xxx.mtc csadm AuthRSASign=paul,{SHA-256}:cs2:cp8:COM1 AuthClrPwd=pauline,1234hello DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message
Note	If the user's private RSA key is stored on a smart card, the user will be prompted at the PIN-Pad to insert his smart card and enter the PIN. The PIN-Pad has to be connected to a serial line of the computer where the CSADM tool is running.

4.10.2 AuthClrPwd

With this command a user with the mechanism 'Clear Password' authenticates a single command.

The following steps are performed during command execution:

1. Command data and plain password are sent to the CryptoServer.
2. The CryptoServer compares the user's password with the stored password from its user database.
3. If the comparison has been successful (and if the necessary authentication state is achieved), the CryptoServer will execute the command.

Syntax	csadm [Dev=<device>] AuthClrPwd=<user>,<password> ... ... [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <user> user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended!); otherwise: user's password <command> command which has to be authenticated
Examples	■ csadm AuthClrPwd=paul,swordfish DeleteFile=xxx.mtc ■ csadm AuthClrPwd=paul,ask DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.10.3 AuthSha1Pwd

With this command a user with the authentication mechanism 'SHA1 hashed Password' authenticates a single command.

The following steps are performed during command execution:

4. A challenge value is requested from the CryptoServer.
5. A SHA-1 hash value is calculated over the user's password, the command data and the challenge value.
6. Command data and hash value are sent to the CryptoServer.
7. The CryptoServer re-calculates the SHA-1 hash via the user's password (taken from the user database), the command data and the challenge value and compares it with the given hash.
8. If the comparison has been successful (and if the necessary authentication state is achieved), the CryptoServer will execute the command

Syntax	csadm [Dev=<device>] AuthSHA1Pwd=<user>,<password> ... ... [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <user> user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's password <command> command which has to be authenticated
Examples	■ csadm AuthSHA1Pwd=paul,swordfish DeleteFile=xxx.mtc ■ csadm AuthSHA1Pwd=paul,ask DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.10.4 AuthRSASC

With this command a user with the mechanism 'RSA Smart Card' authenticates a single command.

For this authentication mechanism, the PIN-Pad (including smart card reader) has to be connected directly to CryptoServer's serial port COM2.

The following steps are performed during command execution:

1. The command data is sent to the CryptoServer.
2. The CryptoServer generates a challenge value.
3. The user signs the challenge value with his smart card (which has to be inserted into the smart card reader) on which his/her private RSA key is stored.
4. The CryptoServer verifies the signature with the public part of the user's RSA key from its user database.
5. If the verification has been successful (and if the necessary authentication state is achieved), the CryptoServer will execute the command.

Syntax	csadm [Dev=<device>] AuthRSASC=<user> ... ... [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <user> user name <command> command which has to be authenticated
Example	csadm AuthRSASC=paul DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message
Note	The user will be prompted at the PIN-Pad to insert his smart card and enter the PIN. The PIN-Pad has to be connected to the second serial line of the CryptoServer.

4.10.5 AuthHMACPwd

With this command a user with the authentication mechanism 'HMAC Password' authenticates a single command.

The following steps are performed during command execution:

1. A challenge value is requested from the CryptoServer.
2. A HMAC value is calculated over the command data and the challenge value using the user's password as the HMAC key.
3. Command data and hash (HMAC) value are sent to the CryptoServer.
4. The CryptoServer re-calculates the HMAC value via the user's password (taken from the user database), the command data and the challenge value and compares it with the given hash.
5. If the comparison has been successful (and if the necessary authentication state is achieved), the CryptoServer will execute the command

Syntax	csadm [Dev=<device>] AuthHMACPwd=<user>,<password> ... ... [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <user> user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's password In case the user uses a non-default hash algorithm for his HMAC authentication mechanism, the name of this hash algorithm, put into curly braces { }, has to be appended or prepended to the password (see table with user properties in 4.8.17 above and example below). <command> command which has to be authenticated
Examples	■ csadm AuthHMACPwd=paul,swordfish DeleteFile=xxx.mtc ■ csadm AuthHMACPwd=paul,{SHA-512}ask DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.10.6 AuthECDSA

With this command a user with the authentication mechanism 'ECDSA Signature' authenticates a single command.

The following steps are performed during command execution:

1. A challenge value is requested from the CryptoServer.
2. Command data and challenge value are signed using the private part of the user's ECDSA key.
3. Command data and signature are sent to the CryptoServer.
4. The CryptoServer verifies the signature using the public part of the user's ECDSA key from its user database.
5. If the verification has been successful (and if the necessary authentication state has been achieved), the CryptoServer will execute the command

Syntax	csadm [Dev=<device>] [Password=<password>] … … AuthECDSA=<user>,<keyspec> [<*further Authentication*>] <command>
Parameters	<device> device specifier (see 4.1.2) <password> if user's ECDSA key is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <user> user name <keyspec> private part of the user's ECDSA key (smart card or key file, see 4.1.3) In case the user uses a non-default hash algorithm for his ECDSA Signature authentication mechanism, the name of this hash algorithm, put into curly braces { }, has to be appended or prepended to the keyspecifier (see table with user properties in 4.8.17 above and example below). <command> command which has to be authenticated
Examples	csadm AuthECDSA=paul,:cs2:cp8:COM1 DeleteFile=xxx.mtc csadm AuthECDSA=paul,{SHA-512}:cs2:cp8:COM1 AuthClrPwd=pauline,1234hello DeleteFile=xxx.mtc
required state	*operational*
Output	none on success, or error message
Note	If the user's private ECDSA key is stored on a smart card, the user will be prompted at the PIN-Pad to insert his smart card and enter the PIN. The PIN-Pad has to be connected to a serial line of the computer where the CSADM tool is running.

4.10.7 ShowAuthState

CryptoServer's current authentication state is displayed.

For the meaning of the authentication state, see 2.6.

Syntax	csadm [Dev=<device>] ShowAuthState
Parameter	<device> device specifier (see 4.1.2)
required state	*operational*
Authentication	none
Output	**current AUTH state: 03000012**
Note	The authentication state is displayed as the sum of permissions of all users who are currently statically logged into the CryptoServer (see *Login*, chapter 4.10.8). **Example:** Two users are statically logged in. Permission of user1: 02000010 Permission of user2: 01000002 Resulting Auth.state: 03000012

4.10.8 Login

With this command a user can log in statically. This means that after a successful authentication of this *Login* command, CryptoServer's authentication state will be augmented by the permissions of the user. (For the meaning of the authentication state, see 2.6.)

Example:

Authentication state before user's static login:	01000000
Permissions of the user:	02000001
Authentication state after user's static login:	03000001

- *The* Login *command can only be executed by users with the appropriate permission (i. e. the corresponding property flag 'allow_login' was set when the user was created with* AddUserXXX *command, see 4.8.17).*
- *The reached authentication state remains unchanged until the* Logoff *command is executed (see chapter 4.10.9).*
- *The authentication state can be increased if more users log in statically.*

All commands (of the same command group, i. e. requiring the same authentication state) issued between the *Login* and a *Logoff* do not require an explicit authentication.

In no case the Login command should be used if it cannot be made sure that *- during the time when the user is statically logged in -* ***no unauthorized user is able to send commands to the CryptoServer****. (This is e. g. important when the CryptoServer is connected to a LAN.)*

For security reasons the main administrator ADMIN is not allowed to use static login.

In any case the Logoff *command must be executed immediately after the static login is no longer needed.*

Syntax	csadm [Dev=<device>] <*Authentication*> Login
Parameter	<device> device specifier (see 4.1.2)
required state	*operational*
Authentication	any
Output	none on success, or error message
Note	If a user is logged in, a second login will have no effect.

4.10.9 Logoff

With this command all users are logged off from the CryptoServer, i. e. CryptoServer's authentication state is set to '00000000'. (For the meaning of the authentication state, see 2.6)

Syntax	csadm [Dev=<device>] Logoff
Parameter	<device> device specifier (see 4.1.2)
Authentication	none
required state	*operational*
Output	none on success, or error message

4.11 Secure Messaging

With Secure Messaging, commands to and from the CryptoServer will be encrypted and integrity protected using a 32 bytes AES session key which was exchanged when opening the session.

See also 2.7 for a description of CryptoServer's Secure Messaging concept.

A session will automatically be closed when the command execution finishes and CSADM ends. Therefore no 'CloseSession'-command is provided in CSADM.

4.11.1 SessionRSA

With this command a user whose authentication mechanism uses an RSA key pair (i. e. the 'RSA Signature' mechanism or the 'RSA Smart Card' mechanism) opens a session for Secure Messaging.

Syntax	csadm [Dev=<device>] [<*Authentication*>] [Password=<password>] … SessionRSA=<user>,<keyspec> <commands>
Parameters	<device> device specifier (see 4.1.2) <password> if the (private part of) user's RSA key (referenced in keyspec parameter) is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <user> user name <keyspec> (private part of) user's RSA key *(see 4.1.3 for key specifiers)* <commands> commands which should be executed within the session
required state	*operational*
Authentication	depending on the *user's property-flag* (see 4.8.17): ■ *sm*: authentication optional (not required); if authentication is wanted the command can be authenticated by any user(s) ■ *sma*: authentication required by the same user who opens the session (according to the user's authentication mechanism, see chapter 4.10).
Output	none on success or error message
Note	■ If a user has authenticated the session his permissions are granted to the whole session (to all commands following the *SessionRSA* command). ■ If the private RSA key of the user is stored on a smart card, the user will be prompted at the PIN-Pad to insert his smart card and enter the PIN. The PIN-Pad has to be connected to a serial line of the computer where the CSADM tool is running. ■ If authentication with a RSA key from a smart card is used the user(s) will be prompted twice at the PIN-Pad: Once for the authentication and a second time for establishing the session.

4.11.2 SessionEC

With this command a user who uses the 'ECDSA Signature' authentication mechanism opens a session for Secure Messaging.

Syntax	csadm [Dev=<device>] [<*Authentication*>] [Password=<password>] … SessionEC=<user>,<keyspec> <commands>
Parameters	<device> device specifier (see 4.1.2) <password> if the (private part of) user's ECDSA key (referenced in keyspec parameter) is given in an encrypted key file: password of encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: password of key file *(see 4.1.3 for encrypted key files)* <user> user name <keyspec> (private part of) user's ECDSA key *(see 4.1.3 for key specifiers)* <commands> commands which should be executed within the session
required state	*operational*
Authentication	depending on the *user's property-flag* (see 4.8.17): ■ *sm*: authentication optional (not required); if authentication is wanted the command can be authenticated by any user(s) ■ *sma*: authentication required by the same user who opens the session (according to the user's authentication mechanism, see chapter 4.10).
Output	none on success or error message
Note	■ If a user has authenticated the session his permissions are granted to the whole session (to all commands following the *SessionEC* command). ■ If the private ECDSA key of the user is stored on a smart card, the user will be prompted at the PIN-Pad to insert his smart card and enter the PIN. The PIN-Pad has to be connected to a serial line of the computer where the CSADM tool is running. ■ If authentication with a ECDSA key from a smart card is used the user(s) will be prompted twice at the PIN-Pad: Once for the authentication and a second time for establishing the session.

4.11.3 SessionPwd

With this command a user with 'Clear Password' or 'SHA-1 Hashed Password' authentication mechanism may open a session for Secure Messaging.

Syntax	csadm [Dev=<device>] [<*Authentication*>] ... ... SessionPwd=<user>,<password> <commands>
Parameters	<device> device specifier (see 4.1.2) <user> user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's password <commands> commands which should be executed within the session
required state	*operational*
Authentication	depending on the *user's property-flag* (see 4.8.17): ■ *sm*: authentication optional (not required); if authentication is wanted the command can be authenticated by any user(s). ■ *sma*: authentication required by the same user who opens the session (according to the user's authentication mechanism, see chapter 4.10).
Output	none on success or error message
Note	If a user has authenticated the session, his permissions are granted to the whole session (to all commands following the *SessionPwd* command).

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's password separately (i. e. a request 'Enter Passphrase:' follows in one of the next command lines) and hide the entrance on the monitor by the display of default characters.

4.11.4 SessionHMAC

With this command a user who uses the 'HMAC Password' authentication mechanism may open a session for Secure Messaging.

Syntax	csadm [Dev=<device>] [<*Authentication*>] ... ... SessionHMAC=<user>,<password> <commands>
Parameters	<device> device specifier (see 4.1.2) <user> user name <password> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: user's password <commands> commands which should be executed within the session
required state	*operational*
Authentication	depending on the *user's property-flag* (see 4.8.17): ■ *sm*: authentication optional (not required); if authentication is wanted the command can be authenticated by any user(s). ■ *sma*: authentication required by the same user who opens the session (according to the user's authentication mechanism, see chapter 4.10).
Output	none on success or error message
Note	If a user has authenticated the session, his permissions are granted to the whole session (to all commands following the *SessionHMAC* command).

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the user's password separately (i. e. a request 'Enter Passphrase:' follows in one of the next command lines) and hide the entrance on the monitor by the display of default characters.

4.11.5 SessionDH

This command opens a session for Secure Messaging using the Diffie-Hellman key agreement (according to [PKCS#3]).

Syntax	csadm [Dev=<device>] [<*Authentication*>] SessionDH=<size> <commands>	
Parameters	<device>	device specifier (see 4.1.2)
	<size>	size of the Diffie-Hellman parameters (prime) in bits, valid values are 512, 1024 and 2048
	<commands>	commands which should be executed within the session
required state	*operational*	
Authentication	authentication optional (not required)	
Output	none on success or error message	
Note	If a user has authenticated the session, his permissions are granted to the whole session (to all commands following the *SessionDH* command).	

4.11.6 SessionECDH

This command opens a session for Secure Messaging using the Anonymous Elliptic Curve Diffie-Hellman mechanism with counter and cofactor described in section 6.2 of [ANSI-X9.63].

Syntax	csadm [Dev=<device>] [<*Authentication*>] SessionECDH=<curvename> <commands>
Parameters	<device> device specifier (see 4.1.2) <curvename> name of elliptic curve to be chosen for EC key generation (see section 8) <commands> commands which should be executed within the session
required state	*operational*
Authentication	authentication optional (not required)
Output	none on success or error message
Note	If a user has authenticated the session, his permissions are granted to the whole session (to all commands following the *SessionECDH* command).

4.12 Administration of the CryptoServer LAN

This command group explicitly addresses the CryptoServer LAN (communication unit); commands will not be forwarded to a CryptoServer PCI (PCI card with the 'real' security module). Instead they will be processed by the control module of the TCP-Server ('csxlan'-daemon) and responded in the same way a firmware module would respond to a command.

These commands are used to administrate a CryptoServer LAN remotely (e. g. to get/put its configuration).

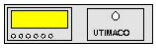
Since the commands of this group are not directed to any CryptoServer (PCI) but only to the CryptoServer LAN, the presence, mode or state of eventually underlying CryptoServers are not relevant for their performance. For the same reason, none of these commands have to be authenticated using the CryptoServer's authentication mechanism.

Nevertheless some commands of this group are protected against unauthorized use with an own authentication mechanism: these commands have to be authenticated with the root password of the CryptoServer LAN. See also [CSLANUserManual].

4.12.1 CSLGetConnections

All current connections to the CryptoServer LAN are listed.

Syntax	csadm [Dev=<device>] CSLGetConnections
Parameter	<device> device specifier (see 4.1.2); "PCI:" addressing is not allowed
Authentication	none
Output	**current connections to csxlan** **# ip address port protocol** **1: 192.168.4.122 1131 TCP** **2: 192.168.4.098 2563 TCP** **3: 192.168.4.122 1137 TCP**
Note	■ The following information is shown: ip-address: IP-address of the client PC port: port on the client PC used to open the connection protocol: used protocol (either UDP or TCP) ■ Up to 10.000 connections can be established to one CryptoServer LAN at the same time.



With the CryptoServer LAN this command will be performed by choosing the menu point
LAN Box administration ➔ State ➔ List Clients

4.12.2 CSLScanDevices

The *CSLScanDevices* command returns the internal configuration of one or more CryptoServer LAN as set up in the configuration file '/etc/csxlan.conf' on each CryptoServer LAN (see [CSLANUserManual]). If the command is called using the dedicated IP-address (as 'Dev='-parameter) of a CryptoServer LAN, only the configuration of this single CryptoServer LAN will be shown.

However, *CSLScanDevices* can also be used to scan the whole network for all CryptoServer LAN currently available. Therefore this command has to be sent as 'Multicast Message' by using the special IP address 'UDP:224.1.0.1' as device parameter (Dev=). It will be responded by every CryptoServer LAN which has been set up to respond to Multicast Messages (see [CSLANUserManual]).

CSLScanDevices *does only find such CryptoServer LAN which have been configured to respond Multicast Messages.*

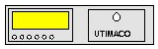
Syntax	csadm [Dev=<device>] CSLScanDevices
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <device> can be for example 'UDP:224.1.0.1' (Multicast) if the whole network shall be scanned.
Authentication	none
Output	**CSLAN ip address port prot device** **----------** **1 192.168.4.201 288 TCP PCI:/dev/cs2a** **1 192.168.4.201 289 TCP TCP:192.168.10.50** **1 127.0.0.1 288 UDP PCI:/dev/cs2a** **1 224.1.0.1 288 UDP PCI:/dev/cs2a** **2 192.168.10.50 288 TCP PCI:/dev/cs2a** **2 192.168.10.50 57 TCP PCI:/dev/cs2a** **2 127.0.0.1 288 UDP PCI:/dev/cs2a** **2 224.1.0.1 288 UDP PCI:/dev/cs2a** **3 192.168.4.127 288 TCP PCI:/dev/cs2a** **3 192.168.4.127 289 TCP PCI:/dev/cs2b** **3 224.1.0.1 288 UDP PCI:/dev/cs2a**

Note	On a CryptoServer LAN the real CryptoServer PCI is accessible by setting up several parameters: ■ ip address: interface address (either the TCP/IP address of the CryptoServer LAN, the localhost address '127.0.0.1' or the Multicast address '224.1.0.1') ■ port: port the TCP-Server (daemon) listens to (default: 288, each device can be set up with multiple ports) ■ protocol: either TCP or UDP (UDP has a packet size limitation) ■ device: either a local CryptoServer (PCI) or a second CryptoServer LAN (TCP)

4.12.3 CSLGetVersion

The version number of the TCP-Server ('csxlan'-daemon) is shown.

Syntax	csadm [Dev=<device>] CSLGetVersion
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed.
Authentication	none
Output	**csxlan 1.0.8 (13.06.2003)**

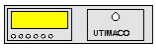


With the CryptoServer LAN this command will be performed by choosing the menu point
LAN Box administration ➔ State ➔ Show Version

4.12.4 CSLGetLogFile

With this command the log file from the CryptoServer LAN can be retrieved.

Syntax	csadm [Dev=<device>] CSLGetLogFile[=<fileno>]
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <fileno> number of the log file on the CryptoServer LAN. <fileno> must be between 1 and 9. If omitted the first log file (number 0) is retrieved.
Authentication	none
Output	content of log file on success, or error message
Note	■ Use '>' to dump the output into a file on the local computer. **Example:** csadm CSLGetLogFile > c:\temp\csxlan.log ■ The log file is formatted according to the UNIX style. Use an appropriate editor (e. g. WordPad) on a Windows system. ■ The contents of the log file depend on the trace level setting of the CryptoServer LAN. See [CSLANUserManual] for a description how to set up logging on the CryptoServer LAN.

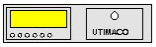


On a CryptoServer LAN the log files can be exported to a floppy by choosing the menu point
LAN Box administration ➔ Diagnostic ➔ Export Trace File

4.12.5 CSLGetConfigFile

With this command the configuration file ('/etc/csxlan.conf') of the CryptoServer LAN can be retrieved.

Syntax	csadm [Dev=<device>] CSLGetConfigFile
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed
Authentication	none
Output	contents of configuration file on success, or error message
Note	■ Use '>' to dump the output into a file. **Example:** csadm GetConfigFile > c:\temp\csxlan.conf ■ The configuration file is formatted according to the UNIX style. Use an appropriate editor (e. g. WordPad) on a Windows system. ■ See [CSLANUserManual] for a description of how to configure the CryptoServer LAN.

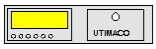


With the CryptoServer LAN this command will be performed by choosing the menu point
`LAN Box administration` ➔ `Configuration` ➔ `Export csxlan.conf`

4.12.6 CSLPutConfigFile

This command imports a new configuration file into a CryptoServer LAN.

Syntax	csadm [Dev=<device>] CSLPutConfigFile=<password>,<file>
Parameters	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <password> root password of the CryptoServer LAN ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password <file> configuration file to be imported
Authentication	special password authentication using the root password of the CryptoServer LAN
Output	none on success, or error message
Note	■ Independently from the given filename the configuration file will be imported as '/etc/csxlan.conf'. ■ The imported file may also be formatted according to the Windows style. On the CryptoServer LAN the file will be reformatted to the UNIX format style. ■ See [CSLANUserManual] for a description of how to configure the CryptoServer LAN.



With the CryptoServer LAN this command will be performed by choosing the menu point
LAN Box administration ➔ Configuration ➔ Import csxlan.conf

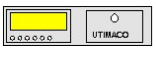
4.12.7 CSLSetTracelevel

This command sets the trace level (== log level) of a CryptoServer LAN. Depending on the trace level more or less information will be written into the log file ('/var/log/csxlan.log'). Each level is independent from the others (the *Verbose* level does not include the *Info* level).

Any 'added' combination of the following values can be used:

level	***value***	***description***
Info	0x80	informational messages like connection establishment, termination and the execution of functions of the control module are written to the log file
Verbose	0x40	details about the state machine are logged
Packet Data	0x20	the content of request and reply packets is logged (max. 256 bytes per packet)

Syntax	csadm [Dev=<device>] CSLSetTraceLevel=<password>,<level>
Parameters	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <password> root password of the CryptoServer LAN: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password <level> desired trace level, may be any (added) combination of: Info: 0x80 Verbose: 0x40 Packet_Data: 0x20
Authentication	special password authentication using the root password of the CryptoServer LAN
Output	none on success or error message
Note	The new trace level is only a temporary setting. The next time CryptoServer LAN (re)starts, the level which is put down in the configuration file ('/etc/csxlan.conf') will be used again.

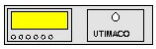


With the CryptoServer LAN this command will be performed by choosing the menu point
`LAN Box administration ➔ Diagnostic ➔ Trace Level`

4.12.8 CSLShutdown

This command shuts down the CryptoServer LAN as the Linux command 'shutdown -h' would do at the local console. All programs and services will be stopped, all devices will be unmounted and the system will be put into 'RunLevel 0'. The CryptoServer LAN can then be powered off.

Syntax	csadm [Dev=<device>] CSLShutdown=<password>
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <password> root password of the CryptoServer LAN: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password
Authentication	special password authentication using the root password of the CryptoServer LAN
Output	none on success or error message

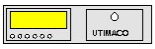


With the CryptoServer LAN this command will be performed by choosing the menu point
LAN Box administration ➔ Shutdown

4.12.9 CSLReboot

This command reboots the CryptoServer LAN the same way the Linux command 'shutdown -r' would do at the local console. All programs and services will be stopped, all devices will be unmounted and the system will rebooted again.

Syntax	csadm [Dev=<device>] CSLReboot=<password>
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed. <password> root password of the CryptoServer LAN: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password
Authentication	special password authentication using the root password of the CryptoServer LAN
Output	none on success or error message



With the CryptoServer LAN this command will be performed by choosing the menu point
`LAN Box administration ➔ Restart`

4.12.10 CSLGetTime

This command reads the Real Time Clock of the CryptoServer LAN (not the clock of the CryptoServer HSM) and outputs the date and time.

Syntax	csadm [Dev=<device>] CSLGetTime
Parameter	<device> device specifier (see 4.1.2). "PCI:" addressing is not allowed.
Authentication	none
Output	date and time of the CryptoServer LAN

4.12.11 CSLSetTime

This command sets the Real Time Clock of the CryptoServer LAN (not the Clock of the CryptoServer HSM).

Syntax	csadm [Dev=<device>] CSLSetTime=<password>,<time>
Parameter	<device> device specifier (see 4.1.2). “PCI:” addressing is not allowed. <password> root password of the CryptoServer LAN: ■ for hidden password entry: string ‘ask’ (see 4.1.4; hidden password entry is strongly recommended!); ■ otherwise: root password <time> ‘YYYYMMDDHHMMSS’ or ‘SYSTEM’
Authentication	special password authentication using the root password of the CryptoServer LAN
Output	none on success or error message
Note	■ The time has to be given as digit string: YYYYMMDDHHMMSS where YYYY=year, MM=month, DD=day, HH=hour, MM=minute, SS=second ■ If SYSTEM is given as argument the system time of the Host PC is used.

4.12.12 CSLGetSerial

This command reads and outputs the serial number of the CryptoServer LAN (not the serial number of the CryptoServer HSM).

Syntax	csadm [Dev=<device>] CSLGetSerial
Parameter	<device> device specifier (see 4.1.2). “PCI:” addressing is not allowed.
Authentication	none
Output	serial number of the CryptoServer LAN (if available)

4.12.13 CSLGetLoad

This command reads and outputs the working load of the CryptoServer in percent.

Syntax	csadm [Dev=<device>] CSLGetLoad
Parameter	<device> device specifier (see 4.1.2). “PCI:” addressing is not allowed.
Authentication	none
Output	Working load of the CryptoServer. If more than one CryptoServer HSM is installed inside the CryptoServer LAN, for every CryptoServer the working load is printed.

The information of the working load of the CryptoServer is read from the display module of the CryptoServer LAN. If the display module is busy (i.e. an operator is working at the display) this information is not available and the command does not output any value.

The value retuned by the command is not the working load at that time but is an average value of the last 60 seconds. It is recalculated and updated every 5 seconds.

4.13 Initialization Key and User Key Management

In this chapter CSADM commands for generation, administration and backup of the *Initialization Key* or other user's authentication keys (or tests keys) are described. The generated keys may be stored either on a smartcard or in a key file.

4.13.1 GenKey

This command generates a key file '`*.key`' which contains a RSA or ECDSA key pair of given size (for key files see 4.1.3). This can be used e. g. to generate an *Initialization Key*, any user's RSA or ECDSA key, or any test key.

It can be chosen if the generated key file is in plaintext or encrypted (see 4.1.3 for encrypted key files).

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm [NewPassword=<password>] [KeyType=RSA] GenKey=<file>[,<bitsize>[,<keyname>]] csadm [NewPassword=<password>] KeyType=EC GenKey=<file>,<curve>[,<keyname>]
Parameters	<password> password to protect encrypted key file: ■ for hidden password entry: string 'ask' (see 4.1.4; hidden password entry is strongly recommended); ■ otherwise: new password of key file <file> name of the key file (with path, filename extension '*.key') <bitsize> bit length of the generated RSA key (512 <= bitsize <= 8192) (default value: 1024 bit) <curve> name of the elliptic curve to be used for ECDSA key generation (see section 8) <keyname> key name
Examples	csadm GenKey=C:\my_keys\testkey1.key,2048,init-testkey1 csadm KeyType=EC GenKey=C:\my_keys\testkey2.key,secp256k1,key2
Output	none on success or error message

Note	■ If the 'NewPassword' parameter is given, the command will generate an encrypted key file, protected by the given password (see 4.1.3 for encrypted key files). If the parameter is omitted, the command will generate a plaintext key file. ■ As default a RSA key will be generated. If an ECDSA key shall be generated, the parameter 'KeyType=EC' has to be given. ■ For the RSA key length the size 1024 bits will be taken as default value. ■ Apart from test purposes, it is strongly recommended not to use RSA keys of size 512 bits. ■ The parameter 'keyname' can be used to assign a name to the RSA or ECDSA key. If the parameter is not given, the key name will be left empty.

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the password separately (i. e. a request 'Enter Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.13.2 SaveKey

This command reads the *Initialization Key* or any other user's authentication key or its public part from one storage medium and stores it to another.

The key may be read from a key file, from a smartcard (e.g. administrator smartcard) or from two backup-smartcards.

The key could be transferred either to a keyfile or to a smartcard.

It is neither possible to transfer the secret key out of a smartcard to any other medium, nor to transfer the public key part only to a smartcard.

This command is executed locally without any connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm InitPubKey=<keyspec1> SaveKey=<keyspec2> csadm InitPrvKey=<keyspec1> SaveKey=<keyspec2> csadm InitBckKey=<keyspec1> SaveKey=<keyspec2>
Parameter	<keyspec1> key specifier (see 4.1.3) of user's key (e.g. Initialization Key): • 'InitPubKey=': public part of key • 'InitPrvKey=': private part of key (only key specifier of key file possible) • 'InitBckKey=': XOR-halves of key on back-up smartcards <keyspec2> key specifier (see 4.1.3) for keyfile/smartcard where the key should be copied to (for public key part only keyfile possible)
Examples	csadm InitPubKey=:cs2:cp8:COM1 SaveKey=C:\keys\pubkey1.key csadm InitBckKey=:cs2:cp8:COM1 SaveKey=:cs2:cp8:COM1 csadm InitPubKey=C:\keys\prvkey1.key SaveKey=C:\keys\pubkey1.key csadm InitPrvKey=C:\keys\prvkey1.key#ask SaveKey=:cs2:cp8:COM1
Output	none on success, or error message
Note	• If smartcards are used: A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad's display for instructions on further command processing.

Apart from test environments, it is strongly recommended to store private keys only on smart cards! Only in this case the private key will never leave the secure token and not be used for calculations on the host.

4.13.3 BackupKey

This command saves the Initialization Key or any other user's authentication key in two XOR-halfes on two smartcards for back-up matters. The key must be available as a key file.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm InitPrvKey=<keyfile> BackupKey=<keyspec>
Parameter	<keyfile> file where the key is stored (*.key, see 4.1.3) <keyspec> key specifier of the back-up smartcards (see 4.1.3)
Examples	csadm InitPrvKey=C:\my_keys\prvkey1.key BackupKey=:cs2:acr:COM1 csadm InitPrvKey=C:\my_keys\myprvkey.key#mypassword BackupKey=:cs2:acr:COM1
Output	none on success, or error message
Note	A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad's display for instructions on further command processing.

4.13.4 CopyBackupCard

This command copies one XOR-half of a key-backup (backup of Initialization Key or any other user's authentication key) from one smartcard to another.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm CopyBackupCard=<keyspec>
Parameter	<keyspec> key specifier of the backup-smartcards (see 4.1.3), source and target
Examples	csadm CopyBackupCard=:cs2:acr:COM1
Output	none on success, or error message
Note	A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad's display for instructions on further command processing.

4.13.5 GetCardInfo

The *GetCardInfo* command reads information about keys eventually stored on a smartcard: The info records of the user's authentication key (e.g. Initialization Key) and back-up key will be scanned and output.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm GetCardInfo=<keyspec>
Parameter	<keyspec> key specifier of the smartcard (see 4.1.3)
Examples	csadm GetCardInfo=:cs2:acr:COM1
Output	Example: **RSA-Key: Utimaco Safeware AG / Init-Dev-1-Key** **Key-Backup: (no key)**
Note	A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad's display for instructions on further command processing.

4.13.6 ChangePassword

This command changes the password of an encrypted key file ('`*.key`', see 4.1.3 for encrypted key files).

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm Password=<password1> NewPassword=<password> ChangePassword=<keyfile>
Parameters	<password1> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: old password of key file <password2> for hidden password entry: string 'ask' (see 4.1.4 and below; hidden password entry is strongly recommended); otherwise: new password of key file <keyfile> encrypted key file ('*.key').
Examples	■ csadm Password=swordfish NewPassword=shark ChangePassword=C:\my_keys\mykey1.key ■ csadm Password=ask NewPassword=ask ChangePassword=C:\my_keys\privatekey1.key
Output	none on success or error message
Note	The command can also be used to convert a plaintext key file into an encrypted key file and vice versa: ■ The command can be used to encrypt a key file that has been a plaintext key file before. In this case the parameter 'Password=' has to be omitted and only the 'NewPassword=' parameter has to be given. ■ The command can also be used to decrypt a key file, i. e. to turn an encrypted key file into a plaintext key file. In this case the parameter 'NewPassword=' has to be omitted and only the 'Password=' parameter has to be given.

The usage of hidden password entry is strongly recommended!

If hidden password entry is used, the CSADM will ask for the key file's old and new password separately (i. e. a request 'Enter Passphrase:' respectively 'Enter New Passphrase:' follows in the next command line) and hide the entrance on the monitor by the display of default characters.

4.13.7 ChangePin

This command changes the PIN of a given smart card. A PIN-Pad including smart card reader must be used for this command.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm ChangePIN=<keyspec>
Parameter	<keyspec> specifier for smart card type, reader type and serial port (see 4.1.3)
Example	csadm ChangePIN=:cs2:cp8:COM1
Output	none on success or error message
Note	The PIN-Pad has to be connected to a serial line of that computer where the CSADM tool is running.

Watch the PIN-Pad's display:

- *enter old PIN first,*
- *enter new PIN then and*
- *confirm new PIN.*

4.14 Master Box Key Management

The CryptoServer is able to store up to four Master Box Keys (slot 0..3) to be used by various applications. Each Master Box Key (MBK) can either be a DES key (16 or 24 bytes length) or an AES key (16, 24 or 32 bytes).

A MBK can be generated on the CryptoServer and - splitted into key parts (key shares) - externally stored on two or more smartcards. The key parts of a MBK can be imported into the CryptoServer either from smartcards or they can be manually typed in at the smartcard reader (PIN-Pad). If a new key should be imported into a key slot which already contains a MBK, the old MBK has to be imported too in order to be verified. The CryptoServer compares the existing key with the key to be verified and overwrites the existing key with the new key only if they match.

In addition to the local management of Master Box Keys, where the PIN-Pad has to be directly connected to the CryptoServer, the MBK can also be managed remotely without having direct physical access to the CryptoServer (which perhaps is located in a data center). To protect the key parts during transmission from/to the CryptoServer, they are encrypted with a key encryption key (KEK). This KEK (32 bytes AES) is generated inside the CryptoServer and will be sent encrypted to the host application in a prior step. As the encryption of the KEK will be done by the public part of the user's RSA key (e.g. the Initialization Key of the CryptoServer), this KEK exchange procedure requires a user with "RSA signature" authentication mechanism (like user ADMIN).

This chapter describes how Master Box Keys (MBK) can be managed remotely.

4.14.1 MBKListKeys

This command lists all MBK's currently stored on a CryptoServer.

Syntax	csadm MBKListKeys
required state	*operational*
Authentication	none
Output	*no len type* *0 16 DES (XOR)* *1 24 DES (SHARE)* *3 32 AES (SHARE)*
Note	The CryptoServer is able to store up to four MBKs (in slots 0..3). 'no' gives the slot number. 'len' gives the MBK key size in bytes. 'type' gives the key type (algorithm) and indicates if the MBK was exported in two XOR halves ('XOR') or in more than two key shares ('SHARE').

4.14.2 MBKGenerateKey

This command generates a MBK on the CryptoServer, splits it into n key shares and transmits the KEK-encrypted key shares to the host. The key shares will be decrypted on the host and stored either on n smartcards or in n key files.

Syntax	csadm UserKey=<user>,<userkey> Key=<keyspec> ... ... MBKGenerateKey=<keytype>,<keylen>[,<n>,<k>]
Parameters	<user> name of the user whose authentication key is used to exchange the KEK (only user with authentication mechanism „RSA signature“ is possible, e.g. user ADMIN) <userkey> key specifier of user’s authentication key (either smartcard specifier or file name), see 4.1.3 <keyspec> key specifier where the generated MBK shares should be stored (see 4.1.3): either smartcard specifier and record number (separated by comma), or list of filenames and passwords where required (e.g. file1#pwd1,file2#pwd2). <keytype> key type of MBK: either DES or AES <keylen> key size of MBK: 16, 24 (DES, AES) or 32 bytes (AES only) <n> number of key shares to be generated (default: 2) <k> number of key shares required to recombine key (default: 2)
Examples	■ csadm UserKey=ADMIN,:cs2:cp8:COM3 Key=:cs2:cp8:COM3,1 MBKGenerateKey=DES,24 ■ csadm UserKey=Paul,I:\cs2\keys\paul.key#ask Key=mbk1.key#swordfish,mbk2.key#sesame MBKGenerateKey=AES,32,2,2 ■ csadm UserKey=Paul,I:\cs2\keys\paul.key Key=:cs2:cp8:COM3,15 MBKGenerateKey=AES,32,5,3
required state	*operational*
Authentication	none
Output	none on success or error message
Note	• If smartcards are used (recommended): A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad’s display for instructions on further command processing. • If key files are used: The usage of encrypted key files is strongly recommended! • The newly generated MBK is not stored on the CryptoServer. To store the MBK on the CryptoServer it has to be re-imported to it, see command *MBKImportKey*.

4.14.3 MBKImportKey

This command imports the key shares of a MBK either from two or more smartcards or from two or more key files.

Syntax	csadm UserKey=<user>,<userkey> [OldKey=<keyspec_old>] ... ... Key=<keyspec_new> MBKImportKey=<key_no>	
Parameters	<user>	name of the user whose authentication key is used to exchange the KEK (only user with authentication mechanism „RSA signature“ is possible, e.g. user ADMIN)
	<userkey>	key specifier of user’s authentication key (either smartcard specifier or key file name), see 4.1.3
	<keyspec_old>	key specifier where the old MBK should be loaded from (see 4.1.3): either smartcard specifier and record number (separated by comma), or list of filenames and passwords where required (e.g. file1#pwd1,file2#pwd2)
	<keyspec_new>	key specifier where the new MBK should be loaded from (see 4.1.3 and above)
	<key_no>	slot number on the CryptoServer where the key should be imported to
Examples	■ csadm UserKey=ADMIN,:cs2:cp8:COM3 Key=:cs2:cp8:COM3,1 MBKImportKey=0 ■ csadm UserKey=Paul,I:\cs2\keys\paul.key#mypassword Key=mbk1.key#swordfish,mbk2.key#sesame MBKImportKey=3 ■ csadm UserKey=Paul,I:\cs2\keys\paul.key#ask OldKey=:cs2:cp8:COM3,15 Key=:cs2:cp8:COM3,15 MBKImportKey=2	
required state	*operational*	
Authentication	none	
Output	none on success or error message	
Note	• If smartcards are used: A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad’s display for instructions on further command processing. • If key files arer used: The usage of encrypted key files, and of hidden password entry, is strongly recommended! • The old MBK has to be given if the desired key slot already contains a MBK. In this case the CryptoServer verifies if the existing MBK matches the old key before it overwrites it with the new key. If the key slot is still empty the new key is accepted without any verification.	

4.14.4 MBKCopyKey

This command copies one key share of a MBK from one storage location to another.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm Key=<keyspec_source> MBKCopyKey=<keyspec_target>	
Parameters	<keyspec_source>	key specifier where the key share should be loaded from (see 4.1.3): either smartcard specifier and record number (separated by comma), or list of filenames and passwords where required (e.g. file1#pwd1,file2#pwd2).
	<keyspec_target>	key specifier where key share should be stored (see 4.1.3 and above)
Examples	■ csadm Key= mbk1.key#swordfish MBKCopyKey=:cs2:cp8:COM3,1 ■ csadm Key=:cs2:cp8:COM3,1 MBKCopyKey=mbk2.key#ask	
Output	none on success or error message	
Note	• With this command the storage location of a MBK can be migrated from smartcard to key file or vice versa. • If smartcards are used: A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. Watch the PIN-Pad's display for instructions on further command processing. • If key files arer used: The usage of encrypted key files is strongly recommended! If encrypted key files are used, the usage of hidden password entry is strongly recommended!	

4.14.5 MBKCardInfo

This command lists the contents of a MBK smartcard. Here, every MBK key share is stored in a separate record.

This command is executed locally without a connection to a CryptoServer. Therefore the state or mode of any underlying CryptoServer is irrelevant for the command execution, and no authentication at any CryptoServer is necessary.

Syntax	csadm MBKCardInfo=<keyspec>
Parameters	<keyspec> key specifier of the MBK smartcard (see 4.1.3)
Examples	csadm MBKCardInfo=:cs2:cp8:COM3
Output	REC LEN TYP ALG NAME TIMEGEN K ID HASH --- 01: 24 1 DES DESKEY 22.01.2007 13:52:01 00 00 C3C2A3F1C9E83A6D 02: 16 0 DES 13.04.2007 13:41:58 00 00 6A7D994A662B4D22 03: 32 3 AES Test 13.04.2007 16:17:01 03 02 397620CEB7C61DBE
Note	■ A PIN-Pad including smartcard reader has to be connected to a serial line of that computer where the CSADM tool is running. ■ A MBK smartcard may contain up to 16 records / MBK parts. For each stored key share the following information is given: • REC: record number • LEN: key length (in bytes), • TYP: share type (only relevant if 0 or 1: first or second XOR-half; otherwise: key share), • ALG: MBK algorithm, • NAME: name of MBK • TIMEGEN: date and time of generation • K: number of shares that are necessary to recombine MBK (only relevant if TYP is not 0 or 1) • ID: ID of this share • HASH: check value over MBK (usually: first 8 bytes of ISO-hash MDC2 over MBK; if ID= 0 or 1: first respectively second 8 bytes of ISO-hash MDC2 over MBK)

4.14.6 MBKCardCopy

This command copies the whole content of a MBK smartcard to another.

Syntax	csadm MBKCardCopy=<keyspec>
Parameters	<keyspec> key specifier of the smartcard (see 4.1.3)
Examples	csadm MBKCardCopy=:cs2:cp8:COM1
Output	none on success or error message
Note	If records on the target smartcard already exist they will be overwritten without additional query.

4.15 Miscellaneous Commands

In this chapter various helpful CSADM commands are described.

4.15.1 Sleep

The *Sleep* command delays the further command processing by the time given.

Syntax	csadm Sleep=<time>
Parameter	<time> time delay in seconds.
Example	csadm StartOS Sleep=3 GetState
Output	none
Note	In the example above a delay of 3 seconds is inserted between the execution of the *StartOS* and the *GetState* commands.

4.15.2 Cmd

The *Cmd* command provides a generic command interface. This makes it possible to send a command as a byte stream to any firmware module without having (developed) a special tool on the host–PC.

Syntax	csadm [Dev=<device>] Cmd=<fc>,<sfc>,<$byte_1$>,<$byte_2$>,<$byte_3$>,...
Parameter	<device> device specifier (see 4.1.2) <fc> function code (module ID of the firmware module) (0x00, ..., 0x3FF or 0, ..., 1023) <sfc> sub function code (number of the called function) (0x00, ..., 0xFF, or 0, ..., 255) <$byte_n$> n^{th} data byte (dependent on the command) (0x00, ..., 0xFF or 0, ..., 255)
Example	csadm AuthSHA1Pwd=paul,swordfish Cmd=0x123,0x5,1,2,3,4,5,6,7,8
required state	*operational*
Authentication	depending on the command
Output	depending on the command
Note	The parameters fc, sfc and $byte_n$ can be coded either as decimal or as hexadecimal value (prefixed with "0x").

Although in theory it is possible to send commands of up to 256 KBytes to the CryptoServer using 'Cmd=', there is a restriction in praxis by the maximal command line length that can be entered if running the CSADM tool on a Windows system. If a longer command byte stream shall be executed, use the CmdFile *command, see 4.15.3.*

4.15.3 CmdFile

The *CmdFile* command provides a generic command interface. Unlike the *Cmd* command it reads the input data from a file. The length of the command contained in the file may be up to 256 kBytes.

This file may contain the following characters and strings:

Character	***Name***	***Description***
'#'	hash sign	rest of line is comment
','	comma	separator
' '	space	separator
TAB	tabulator	separator
CRLF	new line	separator
'hex'	tag	interpret all values as hexadecimal from now on, even if '0x' is omitted.
'dec'	tag	return to normal mode (i. e. hexadecimal interpretation requires a leading '0x')
"..."	string	string which can reach the end of the line

Example:	
#	
# demo command file	
#	
0x123	# function code / module ID
0x05	# sub function code
0x00,0,0x00,11	# hexadecimal and decimal notation may be mixed
"Hello World"	# strings are framed with quotation marks
hex 0F,1E,2D,3C,4B,5A,60,59	# leading hex-tag allows omitting of '0x'!
68,77,86,95,A4,B3,C2,D1	# still understood as hexadecimal values
dec	# return to normal notation
11,22,33	# decimal values

Syntax	csadm [Dev=<device>] CmdFile=<file>
Parameters	<device> device specifier (see 4.1.2) <file> file (name and extension of the file do not matter)
Example	csadm AuthSHA1Pwd=paul,swordfish CmdFile=demo.cmd
required state	*operational*
Authentication	depending on the command
Output	depending on the command

4.15.4 CSTerm

This command retrieves messages from a serial port and displays them on the screen. It can be used instead of a terminal program to view the log output generated by the CryptoServer at start-up (see 2.3.3.4). The correct interface parameters (115200 baud, 8 bit, no parity) are set automatically.

This command is executed locally without sending commands to a CryptoServer. It just listens to a serial port.

Syntax	csadm CSTerm=<port>
Parameter	<port> serial port of the computer (e. g. COM1 or /dev/ttyS0)
Example	csadm CSTerm=COM1
Output	messages read from the serial port
Note	The serial port of the CryptoServer has to be connected to a serial port of that computer where the CSADM tool is running.

5 Batteries of the CryptoServer

The CryptoServer contains a battery to ensure that security relevant data are not lost even when the CryptoServer is turned off. If the device is not used over a prolonged period, i. e. during storage or if the computer is turned off, the battery will have a durability of half a year at a minimum. If the CryptoServer is permanently powered on, the battery is not discharged and the lifetime of the battery increases. This battery is called “carrier battery” because it is mounted on the PCI carrier card.

In case that the carrier battery is not replaced early enough, all data inside the CryptoServer will be deleted!

An additional external battery may be connected to the CryptoServer to increase battery lifetime. In this case the carrier battery is used only after the external battery is exhausted.

See [CSInstall-Manual] for how to exchange the battery.

The state of the batteries has to be checked regularly.

5.1 Check State of the Batteries

To check the state of the batteries the command “GetInfo” of the CSADM tool should be used (see 4.6.4). The output of this command contains a line starting with the string “`batt`” which shows the state of the batteries:

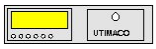
Output	Meaning
`batt ok`	All connected batteries contain sufficient charge.

`batt carrier battery failed.`
or
`batt carrier battery low.`
	The carrier battery of the CryptoServer has low power and must be exchanged.

`batt external battery failed.`
or
`batt external battery low.`
	The external battery has low power and must be exchanged.

`batt carrier battery failed. external battery failed.`
or
`batt carrier battery low. external battery low.`
	The carrier and external batteries both have low power and must be exchanged.

`batt ?`	The state of the batteries could not be determined because of concurrent data transfer to the CryptoServer. In this case (which should happen rarely) repeat the command “GetInfo“ until the battery state is shown.



With the CryptoServer LAN, this command will be performed by choosing the menu point
`CryptoServer administration` ➔ `Generic Commands` ➔ `Show driver info`

6 Typical Administration Tasks

In this chapter the most important administration tasks are explained step by step.

Please keep in mind that for nearly all of these administration tasks the private part of the customer specific *Initialization Key* is needed.

6.1 First Initialization of a New CryptoServer

Please read chapter 3.8 for the CryptoServer delivery concept first.

Precondition:

You have received a new CryptoServer or CryptoServer LAN from Utimaco. This can be either a test system or a production system.

Prior to start operating the CryptoServer please perform the following steps:

6.1.1 CryptoServer Test System

If you have received a CryptoServer test system all firmware had been already loaded and is ready for use. **Utimaco's standard *Initialization Key*** ('Init-Dev-xxx-Key') which is not secret (since its private part is delivered as plain key file) is loaded into the CryptoServer.

1. Perform a *GetState* command (see 4.7.1 or 4.8.1). The CryptoServer should be in *operational* state.
2. Issue a *ListModulesActive* command (see 4.8.7). Check if all necessary firmware modules have been successfully initialized (INIT_OK). Verify the correct versions of the firmware modules.

6.1.2 CryptoServer Production System

If you have received a CryptoServer production system the **customer specific *Initialization Key*** is loaded onto the CryptoServer. No firmware modules are loaded by Utimaco because the private part of the *Initialization Key* is kept secret by the customer. Only the customer is able to administrate the CryptoServer (and to download firmware).

1. Perform a *GetState* command. The CryptoServer should be in *initialized* state and the alarm state should be 'off'.
2. For further initialization of the CryptoServer you need the private part of the customer specific *Initialization Key* (which is usually on the smart card).
3. Additionally you need all firmware modules as '*.mtc' files, properly signed with the customer specific *Initialization Key* (see section 6.3 how to sign firmware modules). Alternatively a firmware package file "*.mpkg" (see 3.7.2) containing the current set of firmware modules can be used.
4. Perform a complete re-initialization of the CryptoServer. Refer to section 6.4 for a detailed description.

6.2 Update Firmware Modules

New firmware modules (e. g. containing new functionality or a bug fix) shall be downloaded onto the CryptoServer, adding to or replacing existing firmware modules.

Precondition:

The new firmware is needed in one of the following forms:

- Either you have received a firmware package file "*.mpkg" from Utimaco, or
- You have received one or more firmware modules (MTC files '*.mtc') from Utimaco.

What to do:

If you use a package file, a firmware update can easily be performed as follows:

1. Load the firmware package file using the *LoadPkg* command (see 4.8.15).

   Doing this, the flags have to be set according to your needs:
   If for instance the firmware modules in the package file shall just be added to the firmware that is already stored inside the CryptoServer, or shall just replace some of the existing firmware modules by other versions, the 'NoDelete'-flag has to be set.

If you want to load a set of firmware modules (MTCs) manually, the following steps have to be performed:

1. Verify whether the CryptoServer is in *operational* state using the *GetState* command.
2. If you have a CryptoServer test system using Utimaco's standard *Initialization Key* you will receive the firmware module(s) as '*.mtc' files ready for download.
3. If you have a CryptoServer production system using a customer specific *Initialization Key* you have to re-sign the new firmware modules with your *Initialization Key* (refer to section 6.3 how to sign firmware modules).
4. For download of the firmware module(s) you need the private part of the *Initialization Key* (usually on the smart card).
5. Download the firmware module(s) using the *LoadFile* command (see 4.8.3).
6. Issue a *Restart* command to the CryptoServer to activate the new firmware module(s).
7. Check if the new firmware module(s) have been properly started using the *ListModulesActive* command (see 4.8.7). Verify the correct version of the firmware module(s) and its/their state (should be INIT_OK). If the new firmware module(s) have not been initialized successfully, the *GetBootLog* command may give detailed information about the problem (see 4.8.8).

6.3 (Re-)Signing Firmware Modules

Firmware modules (actually the data envelope Module Transport Container MTC which contains the executable firmware module, see 3.7.1) have to be signed with the *Initialization Key*. Using a customer specific *Initialization Key* on the CryptoServer requires removing and rebuilding the signature of each new firmware module which you receive from the firmware developer.

Signing or re-signing new firmware modules is not necessary in case you receive the firmware modules in form of a package file "*.mpkg", because in this case the re-signing will be done automatically as part of the *LoadPkg* command during firmware download (see 4.8.15).

Precondition:

You have received one or more firmware modules from Utimaco. They may be in one of the following states:

a) As '*.mmc' files without a MTC signature.

b) As '*.mtc' files signed with Utimaco's standard *Initialization Key*.

c) As '*.mtc' files signed with Utimaco's *Firmware Delivery Key*.

What to do:

1. In case of (a) or (b) the firmware modules should have been transferred to the customer in a secure way (e. g. PGP-encrypted and authenticated).
2. In case of (c) the authenticity of the firmware modules can be verified using the *VerifyMTC* command, see 4.5.3
3. In case of (b) or (c) the old MTC signature has to be removed from each firmware module with the *RemoveMTC* command, see 4.5.2. This creates a '*.mmc' file for each firmware module.
4. Rebuild the signature/MTC of the firmware modules with the correct new *Initialization Key* using the *MakeMTC* command, see 4.5.1. The command creates '*.mtc' files which are ready for download onto the CryptoServer.

6.4 Complete Re-Initialization of the CryptoServer

You want to clear all data inside of a CryptoServer and reload the complete firmware. This may be useful (for example) in the following situations:

- You want to securely clear all secret data inside a CryptoServer.
- You have changed the *Initialization Key*.
- You received a new CryptoServer without any firmware loaded.
- An alarm has occured and has cleared all customer data and firmware modules inside the CryptoServer.

Precondition:

To perform a complete re-initialization, you either need a package file "*.mpkg" containing the current set of firmware modules, or the complete set of all firmware modules as '*.mtc' files.

- In case you use a set of firmware modules, all modules (MTCs) have to be properly signed with the customer specific *Initialization Key* (see section 6.3 how to sign firmware modules).
- In case you use a package file, the contained firmware modules (MTCs) do not necessarily have to be signed, or can even be signed with the wrong key.

Furthermore in both cases the *Initialization Key* is needed.

What to do:

If you use a package file, the re-initialization can easily be performed as follows:

1. Load the package file using the *LoadPkg* command (see 4.8.15). Doing this, set the 'ForceClear' flag.
2. If the previous step has been successfully performed, the CryptoServer is now in *operational* state. You can check if all firmware modules have been started properly using the *ListModulesActive* command (see 4.8.7). If some firmware modules have not been initialized yet, use the *GetBootLog* command (see 4.8.8) to analyze the problem.
3. Optionally (if the CryptoServer is set up the first time): Set the CryptoServer's clock with the *SetTime* command (see 4.8.6).

If you want to re-initialize the CryptoServer manually using a complete set of firmware modules, the following steps have to be performed:

1. If the CryptoServer is not in *initialized* state issue a *ResetToBL* command to switch the CryptoServer into *initialized* state (see 4.6.2).
2. Erase all firmware modules and data using the *BLClear* command (see 4.7.4).
3. Optionally (if the CryptoServer is set up the first time): Set the CryptoServer's clock with the *BLSetRTC* command (see 4.7.8).
4. Load the base firmware modules (SMOS, CMDS, ADM and UTIL) using the *BLLoadFile* command (see 4.7.6).

5. Start the base firmware modules with the *StartOS* command (see 4.7.2).
6. Wait a few seconds, then verify if the CryptoServer is now in *operational* state using the *GetState* command (see 4.8.1).
7. Load all other firmware modules that are necessary for your application with the *LoadFile* command (see 4.8.3).
8. Issue a *Restart* command to activate all new firmware (see 4.6.3).
9. Verify whether the CryptoServer is now in operational state with the *GetState* command and check if all firmware modules have been started properly using the *ListModulesActive* command (see 4.8.7). If some firmware modules have not been initialized yet, use the *GetBootLog* command (see 4.8.8) to analyze the problem.

6.5 Changing the Initialization Key

The customer's *Initialization Key* shall be changed. Therefore its public part which is loaded onto the CryptoServer has to be changed, too.

Preconditions:

- An *Initialization Key* has already been loaded onto the CryptoServer (e. g. by Utimaco Safeware AG). This *Initialization Key* should be replaced.
- The System Administrator has generated an administration smart card with the new *Initialization Key* and a back-up copy for his representative (see 6.6).

What to do:

1. Reset the CryptoServer to boot loader mode using the command *ResetToBL* (see 4.6.2).
2. Execute the *BLChangeInitKey* command (see chapter 4.7.5). Note that the smart card containing the new *Initialization Key* has to be inserted into the smart card reader prior to the smart card with the old *Initialization Key*.
3. Perform a complete re-initialization of the CryptoServer, by use of the new *Initialization Key*. Refer to section 6.4 for a detailed description of this process.

6.6 Generate Your Own Initialization Key

To generate a customer specific *Initialization Key* Utimaco offers a software tool **KeyTool** that is able to

- generate a new RSA key pair (here: *Initialization Key*),
- store the key onto one or more *administration smart cards*,
- store the key in two key halves onto one or more *back-up smart card* pairs.

An *administration smart card* can be used in conjunction with the CSADM tool if the stored key is used as *Initialization Key*. The private part of the RSA key cannot be read out of this smart card.

A *back-up smart card* is used for key storage only. It cannot be used directly for CryptoServer's administration: a back-up smart card contains only one XOR-half of the RSA key, so two back-up cards are necessary to regain the complete key. The key halves can be read out of the card to generate new administration smart cards containing the stored RSA key at a later time. Because two smart cards are needed, this procedure has to be done according to the **2-persons-rule**.

Both types of smart cards (administration smart card, back-up smart card) are protected by a PIN. The Key Tool offers also the functionality to change the PIN

For the generation of a (new) *Initialization Key* it is highly recommended to generate at least one back-up smart card pair and to store all cards securely and seperately.

7 Troubleshooting

7.1 Check Operativeness and State of CryptoServer

This section deals with the situation where it is not known whether the CryptoServer works at all, and in which mode/state it is. The following steps should systematically be performed to check CryptoServer's operativeness and, if possible, to get the CryptoServer working again.

Precondition:

If the CryptoServer is installed on your local computer, make sure that the PCI Driver is running and that the Administration Tool CSADM is installed.

If the CryptoServer is a part of a CryptoServer LAN, make sure that CryptoServer LAN and Client-PC are properly connected to the network (try to 'ping' the LAN box from the Client-PC) and that the Administration Tool CSADM is installed on the Client-PC.

What to do?

1. Perform *GetState* command (see 4.7.1).

Result	Explanation/Reason/Adjustment
state OPERATIONAL	Everything okay, continue with (2).
state INITIALIZED, ALARM = off	CryptoServer is in boot loader mode. ⇒ Start operating system SMOS manually (5).
state INITIALIZED, ALARM = on	An alarm has occurred (and is possibly physically still present) ⇒ see chapter 7.2 for alarm treatment (and 3.3 for more information about CryptoServer alarms).
state neither INITIALIZED nor OPERATIONAL	CryptoServer is not correctly initialized or even defect. ⇒ Please get in contact with manufacturer/Utimaco.
Error B9011xxx, B9015xxx, B9016xxx, B9017xxx or B9021xxx until B9024xxx	CryptoServer's PCI carrier card does not react. ⇒ Try a restart (3).
other errors: B901xxxx or B902xxxx	No connection to CryptoServer / CryptoServer LAN, communication problem, wrong host or device name, problem with network. ⇒ Check parameters. ⇒ Perform ‚ping' at CryptoServer LAN. ⇒ Check state/configuration of the TCP daemon on the CryptoServer LAN.

2. Perform *ListModulesActive* command (see 4.8.7).

Result	Explanation/Reason/Adjustment
All necessary modules are listed and initialized (INIT_OK).	OK. CryptoServer is in *operational mode* and ready to work. End.
Some necessary modules are missing in the given list.	Modules are not loaded onto CryptoServer. ⇒ Check presence of modules with *ListFiles* command. Load missing modules with *LoadFile* and restart CryptoServer. If then modules cannot be started (e. g. wrong signature): ⇒ perform *GetBootLog* and search boot log for errors.
At least one module is not initialized (i. e. not INIT_OK).	Firmware module(s) cannot be started (e. g. module dependencies cannot be resolved). ⇒ perform *GetBootLog* and search boot log for errors

3. Perform *Restart* command (see 4.6.3).

Result	Explanation/Reason/Adjustment
no error	OK ⇒ back to (1)
error 0xB901773x	⇒ Switch CryptoServer off and on again, then back to (1).
other error	CryptoServer does not boot correctly. ⇒ Change to boot loader mode (4).

4. Perform *ResetToBL* command (see 4.6.2).

Result	Explanation/Reason/Adjustment
no error	Ok ⇒ back to (1)
error 0xB9017730	⇒ Switch CryptoServer off and on again, then back to (1).
other error	⇒ Switch CryptoServer's power off and on again, then try again *ResetToBL.* If this does not help: hardware problem ⇒ please contact manufacturer/Utimaco

5. Perform *StartOS* command (see 4.7.2).

Result	Explanation/Reason/Adjustment
no error and the *GetState* command works again	OK ⇒ back to (1)
no error but the *GetState* command does not work	Connect a PC's serial port with a crossed serial cable to the CryptoServer's serial port 1 (at the bracket of the PCI-card). Start a terminal program on the PC (115200 baud, 8 bit, no parity), perform the *Restart* command and watch the terminal output. If there are error messages about firmware modules that are not started: ⇒ Start the module recovery system (6). Otherwise ⇒ Re-initialize the CryptoServer (see chapter 6.4).
any error	There is either no operating system SMOS loaded, or the boot loader is not able to start SMOS. ⇒ Re-initialize the CryptoServer (see chapter 6.4).

6. Perform *ResetToBL* command, and then perform *RecoverOS* command.

Result	Explanation/Reason/Adjustment
no error and the *GetState* command works again	Control the loaded modules with *ListFiles* command and load/replace missing or defect modules if necessary (see chapter 6.4). Then perform *Restart* and go back to (1).
error or the *GetState* command does not work	Try to re-initialize the CryptoServer (see chapter 6.4).

7.2 Alarm Treatment

An alarm can be triggered on the CryptoServer as a result of the following reasons:

- Power is too low
- Power is too high
- Temperature too high (> 66°C)
- Temperature too low (< -13°C)
- Outer foil is broken
- Inner foil is broken
- Invalid (corrupted) Master Key $\mathbf{K_{CS2}}$ (this usually occurs in case of an empty battery)
- External Erase is executed (manually, by a short-circuit of the 'External Erase' pins on the PCI-card)

See chapter 3.3 for a detailed description of the CryptoServer's alarm mechanism.

Some alarm reasons can be removed (e. g. exchange low battery or cool down high tempera¬ture), these are called temporary alarms. Other alarm reasons cannot be removed, these are called permanent alarms. Permanent alarms occur e. g. in case of a damage of the inner or outer tamper protecting foil, whereas usually all other possible alarms are temporary.

The *GetState* command shows the reason of an alarm and if the alarm is still present, see 4.7.1. If the reason for an alarm cannot be removed then please get in contact with the manufacturer/Utimaco, else you can reset the pending alarm state (see below).

Precondition:

An alarm has occurred to the CryptoServer. This will be announced with the *GetState* command (**ALARM: ON**). If *GetState* additionally answers with 'Alarm is present' then the alarm is physically still present. But it is also possible that in the meantime the alarm cause has been removed.

The CryptoServer is necessarily in boot loader mode.

What to do?

1. If the alarm is physically still present: Remove the alarm cause if possible. Execute *GetState* again (see chapter 4.7.1) to check the success. Even if the reason for the alarm has been removed, the alarm state will still be 'ON', but the alarm should no longer be shown as 'present' (only as 'has occurred').
2. If the alarm is still shown as 'present': please contact the manufacturer/Utimaco.
3. Else: Perform the *BLResetAlarm* command (see chapter 0).
4. Execute *GetState* again. The alarm state should now be 'OFF'.
5. Since the alarm has cleared all data and firmware modules, a complete re-initialization of the CryptoServer must be performed now (see chapter 6.4).

8 Built-in Elliptic Curves

The CryptoServer offers a collection of elliptic curves which can be used e.g. for ECDSA key generation. Each curve, given by its elliptic curve domain parameters, can be identified by a name.

The following table lists all built-in elliptic curves domain parameters.

Name(s)	Size	Defined in:
secp112r1	112	[SEC2]
secp112r2	112	[SEC2]
secp128r1	128	[SEC2]
secp128r2	128	[SEC2]
brainpoolP160r1	160	[BP]
brainpoolP160t1	160	[BP]
secp160k1	160	[SEC2]
secp160r1	160	[SEC2]
secp160r2	160	[SEC2]
brainpoolP192r1	192	[BP]
brainpoolP192t1	192	[BP]
NIST-P192 / secp192r1	192	[FIPS186-2], [ANSI-X9.62], [SEC2]
secp192k1	192	[SEC2]
brainpoolP224r1	224	[BP]
brainpoolP224t1	224	[BP]
NIST-P224 / secp224r1	224	[FIPS186-2], [ANSI-X9.62], [SEC2]
secp224k1	224	[SEC2]
brainpoolP256r1	256	[BP]
brainpoolP256t1	256	[BP]
NIST-P256 / secp256r1	256	[FIPS186-2], [ANSI-X9.62], [SEC2]
secp256k1	256	[SEC2]
brainpoolP320r1	320	[BP]
brainpoolP320t1	320	[BP]
brainpoolP384r1	384	[BP]
brainpoolP384t1	384	[BP]
NIST-P384 / secp384r1	384	[FIPS186-2], [ANSI-X9.62], [SEC2]
brainpoolP512r1	512	[BP]
brainpoolP512t1	512	[BP]

Name(s)	Size	Defined in:
NIST-P521 / secp521r1	521	[FIPS186-2], [ANSI-X9.62], [SEC2]
sect113r1	113	[SEC2]
sect113r2	113	[SEC2]
sect131r1	131	[SEC2]
sect131r2	131	[SEC2]
NIST-K163 / sect163k1	163	[FIPS186-2], [ANSI-X9.62], [SEC2]
sect163r1	163	[SEC2]
NIST-B163 / sect163r2	163	[FIPS186-2], [ANSI-X9.62], [SEC2]
sect193r1	193	[SEC2]
sect193r2	193	[SEC2]
NIST-K233 / sect233k1	233	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-B233 / sect223r1	233	[FIPS186-2], [ANSI-X9.62], [SEC2]
sect239k1	239	[SEC2]
NIST-K283 / sect283k1	283	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-B283 / sect283r1	283	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-K409 / sect409k1	409	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-B409 / sect409r1	409	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-K571 / sect571k1	571	[FIPS186-2], [ANSI-X9.62], [SEC2]
NIST-B571 / sect571r1	571	[FIPS186-2], [ANSI-X9.62], [SEC2]

9 References

No.	Title/Company	Doc.-No.
[AIS20]	AIS 20, Version 1: " Functionality classes and evaluation methodology for deterministic random number generators, Version 2.0 of 2.12.1999"; BSI (Bundesamt für Sicherheit in der Informationstechnik - Federal Office for Information Security; Germany) http://www.bsi.bund.de/zertifiz/zert/interpr/ais_cc.htm	
[AIS31]	AIS 31, Version 1: "Functionality classes and evaluation methodology for true (physical) random number generators, Version 3.1 of 25.9.2001"; BSI (Bundesamt für Sicherheit in der Informationstechnik - Federal Office for Information Security; Germany) http://www.bsi.bund.de/zertifiz/zert/interpr/ais_cc.htm	
[ANSI-X9.62]	ANS X9.62-2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA) / ANSI (American National Standards Institute)	
[ANSI-X9.63]	ANSI X9.63-2001: Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography / ANSI (American National Standards Institute)	
[BP]	ECC Brainpool Standard Curves and Curve Generation, v1.0, 19.10.2005, www.ecc-brainpool.org	
[CSAPI]	CryptoServer – Application Interface (CSAPI) / Utimaco Safeware AG	2002-0005
[CSCMDS]	CryptoServer – Firmware Module CMDS - Interface Specification / Utimaco Safeware AG	2002-0008
[CSInstall-Manual]	CryptoServer – Installation Manual / Utimaco SafewareAG	2003-0007
[CSLAN-UserManual]	CryptoServer LAN – User Manual / Utimaco Safeware AG	2003-0002
[FIPS186-2]	FIPS PUB 186-2, Digital Signature Standard / National Institute of Standards and Technology (NIST), January 2000	
[PKCS#1]	PKCS#1: RSA Cryptography Standard v2.1, 14th June 2002 / RSA Laboratories, http://www.rsasecurity.com/rsalabs/pkcs	
[PKCS#3]	PKCS#3: Diffie-Hellman Key Agreement Standard v1.4, 1st November 1993 / RSA Laboratories, http://www.rsasecurity.com/rsalabs/pkcs	
[SEC2]	SEC2: Recommended Elliptic Curve Domain Parameters – Certicom Research – September 20, 2000, Version 1.0	