CryptoServer

Administrator's Guide for CryptoServer in FIPS-Mode

Utimaco Safeware AG

Transaction Security

www.utimaco.com

Imprint

Copyright 2006	**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	info.sp@utimaco.de
Document Number	2004-0002
Version	2.0.1
Status	released
Date	18th December 2006
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 CS** (CryptoServer) if run in FIPS-mode. It should be read carefully by all persons who are allowed to assume the role of an *Administrator* for the CryptoServer.

The cryptographic services that are offered by the CryptoServer are not described in this document: detailed command descriptions e. g. for en- or decrypting services, hashing, key generation and key management can not be found in this Administrator's Guide. Moreover, cryptographic services can not be performed by any CryptoServer's Administrator but only by persons who are allowed to assume the Cryptographic User *role.*
A Guide for Cryptographic Users *of the CryptoServer is provided by document [CSFIPS-UserGuide].*

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 out data, it is encapsulated by metal shells, a special tamper detection foil and potting material. The CryptoServer's physical interface for administration is given over the PCI interface and two serial interfaces.

In FIPS mode, 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 offer internal services or administrative services, others offer cryptographic services. 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 system administrator** is the management of the CryptoServer's firmware modules, user management and the initial process of first personalization of the CryptoServer in order to enter FIPS mode.

*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 system 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:

- the host **PC** with inserted CryptoServer (PCI plug-in card),
- a **PIN-Pad** with integrated smart card reader, and
- the **CSADM tool** (a command line utility provided by Utimaco) which must run on the host PC communicating with the CryptoServer.

See [CSInstall-Manual] for instructions on the hardware installation.

This *Administrator's Guide for CryptoServer in FIPS-mode* 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 the process of CryptoServer's first initialization. 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, modi 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 various modes and *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.
- Chapter 4 gives practical guidance through typical administrative tasks, such as CryptoServer's first personalization in order to enter FIPS-mode, clearing the CryptoServer, quitting error states, 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 5, giving the command descriptions.
- In chapter 5 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 administration command and its execution is given in chapters 5.3-5.12, 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.
- Chapter 6 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.
- Chapter 7 lists all firmware modules that are mandatory for a CryptoServer in approved FIPS-mode.

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 5 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 real security module is located on a printed circuit board and encapsulated by metal shells, a special tamper detection envelope (which is a special foil bearing a flexible printed circuit with a serpentine geometric pattern of conductors) and potting material. This hard, opaque enclosure defines the cryptographic boundary of the module.

The hardware is logically designed as follows:

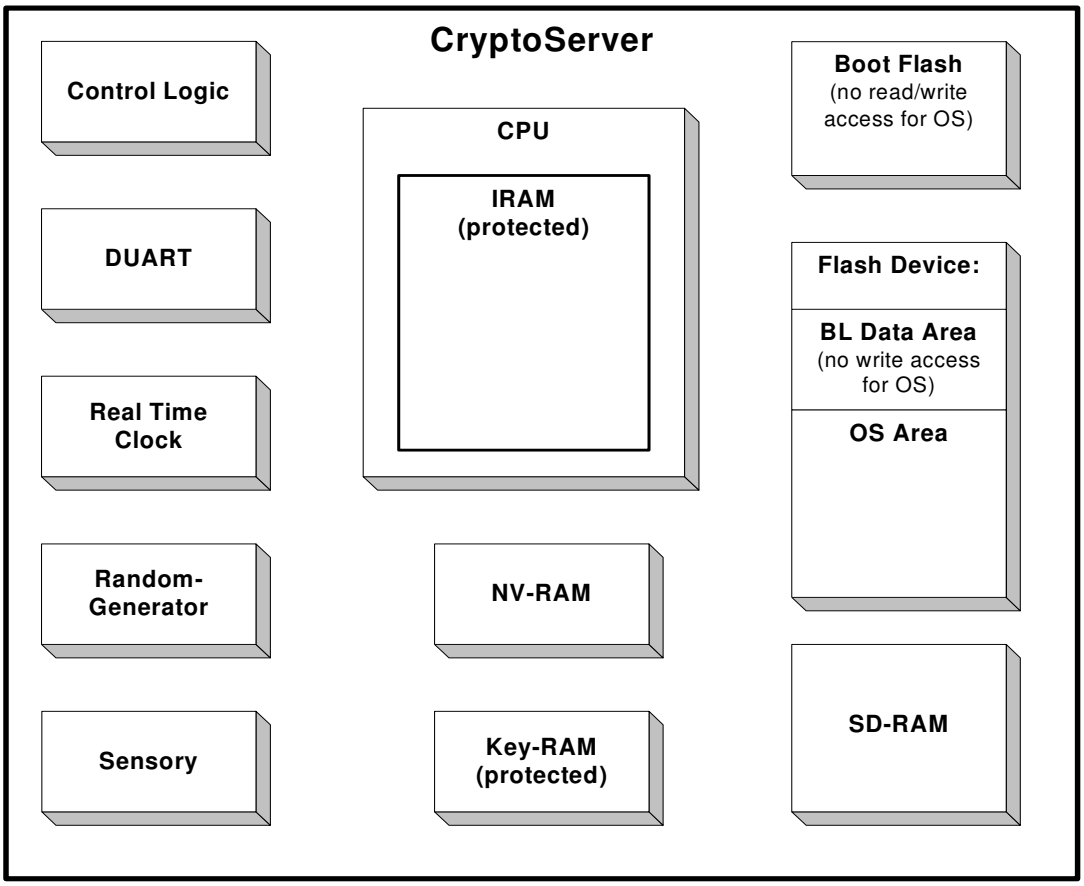

Illustration 2-1: Block Diagram of CryptoServer's Hardware

The several components of the system as shown in the diagram will be described in the following subchapters.

2.1.1 CPU

The CPU (Central Processing Unit) is needed to run the firmware code of the system. Cryptographic algorithms are implemented in software and run on the CPU. The CPU is a modern 32 bit processor (Texas Instruments DSP TMS TMS 320C64xx).

Additionally, the CPU includes memory (*IRAM*, see below). In case of an attack, the *IRAM* will be *actively* erased within a very short time (see chapters 3.3 and 2.1.6.4 for a detailed description).

2.1.2 Control Logic

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

Depending on the global boot state (as described in 3.2) the control logic restricts the read and write access to several memory areas in the flash file (see 2.1.6.2).

2.1.3 Hardware Noise Source for Random Number Generation

The *hardware noise source* is used to seed the CryptoServer's deterministic random number generator:

For random value generation and generation of all cryptographic keys, the CryptoServer relies on an implemented *Deterministic Random Number Generator* (DRNG) compliant with ANSI X9.31 [ANSIX9.31].

Seed and seed key for the DRNG will be generated by the CryptoServer's non-deterministic *True Random Number Generator* (TRNG) which is based on the hardware noise source. The digitalized output of the hardware noise generator will be watched by software: several tests will be done to control the quality of the random bit stream, and a mathematical post-processing will be performed.

2.1.4 Real Time Clock (RTC)

The *real time clock* (RTC) provides date and time (up to milliseconds) to the CryptoServer.

In FIPS-mode the RTC is used for the entries of the alarm log, time log and the temperature log file (adding a timestamp to every entry).

2.1.5 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

If the sensory detects any of these alarm conditions, it will trigger the alarm mechanism to be performed. For more details about CryptoServer's reaction in possible alarm situations see 3.3.

2.1.6 Memory Types

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

2.1.6.1 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).

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.

2.1.6.2 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).

  Additionally, this memory area will contain a copy of the files which have been loaded into the CryptoServer during the production and initialization process (via the appropriate boot loader functionality *LoadFile* which in FIPS mode is not available): the operating system module SMOS 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 (leaving FIPS-mode) 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 operating system) has only read permission.

The boot loader will only write data to this area during the initialization of the CryptoServer. The process to set up and initialize a CryptoServer has to be done before the CryptoServer enters FIPS-mode; this process is described in chapter 4.4.

- 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 will 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. As described in more detail below, in FIPS mode all permanent and encrypted key storage is organized over the firmware module DB which locates the databases with keys and CSPs here in the *OS Area*.

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

2.1.6.3 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* is not deleted actively. Therefore no sensitive plaintext data will be stored in local or global variables. Sensitive data stored in the *SD-RAM* will be encrypted with the CryptoServer's *Master Key* $\mathbf{K_{CS2}}$,

2.1.6.4 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 is going down on power-off, the *IRAM* is erased, too.

2.1.6.5 NV-RAM

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

In FIPS-mode, the NV-RAM will not be used.

2.1.6.6 Key-RAM

The non-volatile I^2C-RAM *Key-RAM* is sensory-protected on runtime and in retention (i. e. if power supply is switched off): in case of an alarm triggered by the sensory, the *Key-RAM* is erased actively within less than 4 msec. 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 is used for the encrypted storage of sensitive data like secret and private keys and other CSPs), 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 5.7.1.*

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* (before entering FIPS-mode).

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 5.7.1.

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 (CryptoServer not in any FIPS Error 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, during the CryptoServer's first personalization at the customer's site, the OS and the basic firmware modules can be loaded by the boot loader. A detailed description of all possible boot loader commands (which offer basic administrative functionality for the CryptoServer's setup and personalization process) is given in section 5.11.[1]

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 – the operating system module SMOS and the firmware modules which provide administrative and cryptographic services to the user of a CryptoServer.

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.*

[1] These commands are only offered as long as the CryptoServer has not entered FIPS-mode yet.

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; these functions are for CryptoServer-internal usage only and not realized as external interface.

2.3.2.2 Further Firmware Modules

The following firmware modules are necessary to operate a CryptoServer in FIPS-mode. Their respective functionality is described in the table below. If one of these firmware modules is not loaded or can not be started, the CryptoServer is either not in FIPS-mode or it will enter FIPS error state. See chapter 7 for a comprehensive list of all necessary firmware modules and their necessary approved versions.

With the exception of the command scheduler module CMDS, the administration module ADM and the module CSI which realizes the cryptographic interface of the CryptoServer, these modules have no external interface but will only be accessed by other firmware modules via internal public C functions. Only CMDS, ADM and CSI have an external interface that can be called from the outside world (via the PCI interface).

Name	Functionality
SMOS	**Small Multitasking Operating System** Operating system of the CryptoServer, see above.
UTIL	**Utilities Module** Only internal functionality: Provides access to the CryptoServer's system clock (RTC) and provides random number generation (with Deterministic RNG) to other firmware modules. Furthermore it provides read access to the Extended Identifier (EID) of the CryptoServer (which will be output with the *GetState* command).

Name	Functionality
CMDS	**Command Scheduler Module** *Internal functionality*: CMDS accepts commands from the outside world of the CryptoServer via the PCI interface and is responsible for processing the protocol stack. It checks the authentication of the commands if necessary (see 2.4.2 for more details). Furthermore CMDS is responsible for processing the *Secure Messaging* layer, see subsection 2.4.3. After that, CMDS is responsible for distribution of received commands to the appropriate firmware modules. *External interface*: Administration of user database (where for each admitted user his user name, permissions and his authentication mechanism is stored); i. e. CMDS offers external commands like *AddUser, ChangeUser, DeleteUser, ….* Here, the sensitive commands offered by CMDS will only be performed if they are authenticated by an user who has assumed the *Administrator* role.
ADM	**Administration Module** Provides an external interface for extended administration, i. e. for loading, replacing and deleting firmware modules, giving RTC access and status and other information. The sensitive commands offered by ADM will only be performed if they are authenticated by a user who has assumed the *Administrator* role.
VDES	**DES Module** Only internal functionality: VDES provides • Triple DES encryption and decryption in ECB or CBC mode, • DES key generation (16 or 24 bytes key length) and • MAC algorithms based on DES (in FIPS mode used: Triple-DES CBC-MAC)
VRSA	**RSA Module** Only internal functionality: VRSA provides • RSA signature generation according to PKCS#1-v2.1 [PKCS#1] (with blinding operation to prevent timing attacks) • RSA signature verification according to PKCS#1-v2.1 [PKCS#1] and • RSA key generation (variable key length).
AES	**AES Module** Only internal functionality: AES provides • AES encryption and decryption in ECB or CBC mode and • AES key generation (128, 192 or 256 bits).
LNA	**Long Number Arithmetic Module** Only internal functionality: LNA provides arithmetic operations with long numbers and generation of (strong) primes for RSA keys (which will e. g. be used by the VRSA module for the key generation)
HASH	**Hash Module** Only internal functionality: Provides SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 hashing algorithm.

Name	Functionality
ECA	**Elliptic Curve Arithmetic Module** Only internal functionality: ECA provides arithmetic operations with points on elliptic curves (which will e. g. be used by the ECDSA module for signature generation)
ECDSA	**ECDSA Module** Only internal functionality: ECDSA provides • ECDSA signature generation • ECDSA signature verification • ECDSA key pair generation (according to [FIPS186-2])
DB	**Database Module** Only internal functionality: DB provides functions to manage data objects like keys or certificates in databases. In these databases data can be stored as clear text or encrypted with the CryptoServer's Master Key $\mathbf{K_{CS2}}$.
ASN1	**ASN1 Parser Module** Only internal functionality: ASN1 contains several functions that support ASN.1 decoding and encoding. (This will be used e. g. to convert a RSA key in PKCS#1 format [PKCS#1] into a RSA key token as it is used by the firmware module VRSA, and vice versa.)
CSI	**Cryptographic Services Interface Module** CSI offers the external command interface for various cryptographic services like key generation, symmetric encryption/decryption (in various DES and AES modes), MAC calculations (based on Triple-DES algorithm), RSA and ECDSA sign data / verify signature, hashing algorithms (SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512) and random number generation (with Deterministic RNG). Additionally, key management functions like import and export of a wrapped key is offered. Apart from the possible export/import of a plain public RSA or ECDSA key, there is no interface for any clear text key import or export. All these commands will only be performed if they are authenticated by a user who has assumed the *Cryptographic User* role.
FIPS140	**FIPS Mode Control Module** This module is only active in *Power-Up Initialization and Self Test 2* state during the CryptoServer's power-up phase. It is responsible to check if all firmware modules that are necessary for FIPS-mode are present, have correctly been started and that none of the power-up self-tests has failed. FIPS140 offers no external interface. Apart from its start-up function (which will be called by SMOS during the power-up phase) it also has no internal interface.

During the process of setup and personalization of the CryptoServer (in order to get a working CryptoServer in FIPS-mode) all these firmware modules have to be loaded. From these modules, the ***basic firmware modules***

- ***SMOS***
- ***UTIL***
- ***CMDS***
- ***ADM***

must be loaded into the CryptoServer in personalization mode by the appropriate boot loader command BLLoadFile*, see 5.11.4. This is necessary to get a failsafe operational CryptoServer with base functionality even in case of an emergency, able to receive further firmware.*
See chapter 4.4 for detailed guidance through the CryptoServer's setup and personalization process.

The basic firmware modules 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 state, with the appropriate ADM command LoadFile*, see 5.7.3). They run on the basis of these basic firmware modules.*

The other modules provide the actual CryptoServer functionality like cryptographic algorithms or key management.

The module dependencies (which module needs functionality from which other module?) can be seen from the following graphics:

CryptoServer - Module Dependencies

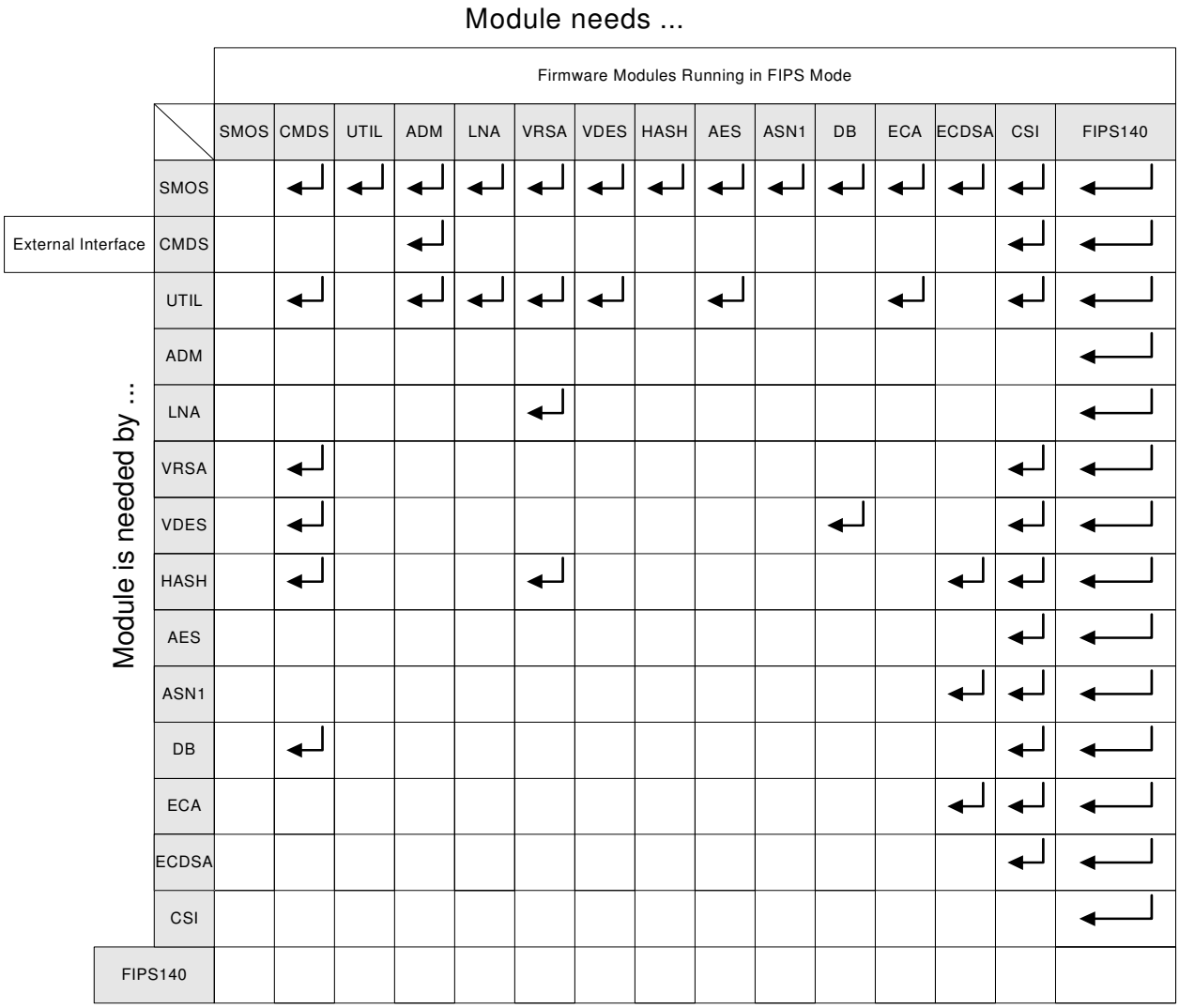

Illustration 2-2: Module dependencies

2.3.3 Boot Process of a CryptoServer in Personalization Mode

*A CryptoServer module that has not entered FIPS-mode (yet) is said to be in **personalization mode**.*

This can be the case after first shipping of the CryptoServer, or after a physical alarm has happened to the module, or after the module has been cleared manually and intentionally (see 4.5).

In this personalization mode the CryptoServer can be set-up and personalized in order to enter FIPS-mode afterwards. In particular, the necessary firmware modules have to be loaded while the CryptoServer is in personalization mode. For this purpose a CryptoServer in personalization mode implements a special boot process. In particular, the boot loader itself offers appropriate interfaces for firmware module loading.

All information given in this section can not be applied to a CryptoServer in FIPS mode!
The boot process described in the following will only be performed by a CryptoServer that is in personalization mode.

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 the 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 hardware 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 erasure and test of the SD-RAM, initialization and test of the random number generator, initialization of the flash file system, alarm handling (if an alarm is present) or extreme temperature handling

if necessary (power-down of the processor if it exceeds the critical limit, in this case the boot process is stopped, see section 3.4). It determines if CryptoServer is in the *manufactured*, *produced*, *initialized* or *defect* state, depending on the result of the self test and the presence of specific keys, see section 3.2. The PCI interface and the serial interface will be initialized.

Then the boot loader offers an open time window for command (which commands are available now depends from the boot loader state, see 3.2). Here basic administrative commands during the CryptoServer's setup and personalization process could be performed.

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 is considered to be in *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 5.11.10 and 5.11.11) 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* state, see section 3.2.4) can be performed.

If the start of SMOS is successful the CryptoServer goes into the global *operational* state and the boot loader terminates. The CryptoServer is still in personalization mode.

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) 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 5.7.8).

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 2.1.6.2) 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 5.6.2 respectively 5.11.11).*

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 5.7.3). After that the CryptoServer can be put again in normal operational mode (running all firmware modules) with a *Restart* command.

2.3.3.6 Boot Process in FIPS-Mode

If the CryptoServer is in FIPS-mode, the boot process is similar to that described above. The most important differences are the following:

1. During the boot loader phase of the boot process, there is no time window for command: Various tests and initialization steps will be performed and, in error free case, at the end of this process the operating system SMOS will be started automatically. Thus no boot loader command can be performed as long as no error has occurred.
2. If during this phase a *FIPS error* is found (if for instance the power-up test for the CryptoServer's deterministic random number generator fails, or if the integrity check for the SMOS module fails), the CryptoServer enters ***Boot Loader Error* state**: The boot loader remains active and a time window for command is opened, but only commands for status requests and loggin information can be performed. See 5.7.
3. When SMOS has started successfully, it performes further tests and initialization steps. In particular it starts and initializes all firmware modules that are found in the flash file. If this was successful, the CryptoServer is considered to be in (FIPS-mode and) normal operational mode, i. e. not in any FIPS error state.
4. If during this phase any *FIPS error* is found, the CryptoServer enters ***OS Error* state**. Examples for FIPS errors include: any cryptographic algorithm test has failed (DES, AES or RSA), or any firmware module that is mandatory in FIPS-mode can not be successfully initialized (e. g. because its software integrity check fails). In *OS Error* state all started software remains active, but only the non-security relevant commands (i. e. those commands that have not to be authenticated) can be performed. See 5.7.
5. The possibilities to watch the boot process and analyze errors are in FIPS-mode the same as described above in 2.3.3.4.
6. The possibilities to recover the back-up copies of the base firmware (see 2.3.3.5 above) are not available in FIPS-mode but only in personalization mode.

2.3.3.7 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 5.6.1) causes a hardware reset. The boot process described in the previous sections is performed. For a CryptoServer in personalization mode, this includes the 10 second time window of the boot loader. If the CryptoServer is in FIPS-mode, this 10 second time window is skipped.
- The *Restart* command (see 5.6.3) 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 5.6.2) 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 the CryptoServer is **in FIPS-mode**, neither the* Restart *nor the* ResetToBL *command is available. In FIPS-mode, only the Reset command can be used (where the 10 second time window for boot loader commands will be skipped).*

*If the CryptoServer is **in personalization mode**, the commands for resetting the CryptoServer should be used according to the following rule:*

- *If boot loader commands shall be performed, use the* ResetToBL *command.*
- *If operational commands shall be performed and/or the OS shall be started, use the* Restart *command.*

2.3.4 CryptoServer's Operator Modi

Depending on the loaded firmware modules, a CryptoServer can either be in FIPS-mode (if the respective FIPS firmware modules are loaded and the FIPS mode is set) or in personalization mode (otherwise).

Additionally, depending on the possible occurrence of (FIPS) errors and the question which firmware is actually active, further modi have to be distinguished.

2.3.4.1 Boot Loader Mode, Operational Mode

Independently from this distinction between FIPS-mode and personalization mode, in a particular moment the CryptoServer can either be in power down mode, in boot loader mode, and in operational mode. This distinction between various modes just refers to the firmware which is active at that special moment: no firmware, only boot loader firmware, or operating system SMOS and further firmware modules.

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*

- **mode = Operational Mode** *or*
- **mode = Boot Loader Mode**

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

For a CryptoServer in FIPS-mode, being in boot loader mode means that the cryptographic module is in FIPS error state. In this case, only very restricted functionality is available: requests for status and logging information are the only services that can be executed in boot loader mode. To leave the boot loader mode / FIPS error state, it is at least necessary to reset the CryptoServer. See section 4.3 how to quit FIPS error state.

For a CryptoServer in personalization mode, the boot loader mode offers access to basic administrative functionality, see 5.11. This is vital for the process to set-up and personalize the CryptoServer in order to enter FIPS-mode afterwards.

2.3.4.2 Possible Modes and States

At the customer's site, the following variations of mode and state of a CryptoServer can occur:

- CryptoServer is in power-down mode (see above).
- CryptoServer is in **FIPS-mode** and **normal operational state** (i. e. no FIPS error has occurred). This includes that the CryptoServer is in **operational mode**. All administrative and cryptographic services are available.
- CryptoServer is in **FIPS-mode**, but in **Boot Loader Error state** (i. e. a FIPS error has been noticed during the first phase of the CryptoServer's boot process). This includes that the CryptoServer is in **boot loader mode**. Only basic status request services are available.
- CryptoServer is in **FIPS-mode**, but in **OS error state** (i. e. a FIPS error has been noticed when the operating system was already up). This includes that the OS is still active; the CryptoServer is thus necessarily in **operational mode**. But only non-sensitive services that have not to be authenticated (like status request services) are available.
- CryptoServer is in **personalization mode** (i. e. not in FIPS-mode!) and in **boot loader mode** (i. e. the bootloader is active). In this mode the CryptoServer is usually ready for the setup and personalization process, see 4.4, but this mode can also occur after an alarm has happened.

- CryptoServer is in **personalization mode** (i. e. not in FIPS-mode!) and in **operational mode** (i. e. the operating system SMOS is already loaded and active). This state and mode usually occurs in the second phase of the personalization process, see 4.4: steps 8-11 will be performed in this state. Which functionality is available depends on the loaded software.

For the respective state indicators, see 4.2. For leaving any FIPS error state, see 4.3.

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. 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 5.

2.4.1 External Interface

Some firmware modules (ADM, CMDS, CSI) offer functionality to the external world, i. e. callable from outside the CryptoServer. Such a 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 (CryptoServer Application Programming Interface)** (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 78, *Illustration 5-1: Command Execution*) 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).

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 administrative standard firmware modules (CMDS, ADM and the boot loader) this job is done by the *CryptoServer Administration Tool CSADM* (which attaches on the CSAPI, see *Illustration 5-1*). This command line utility will be described extensively in chapter 5.
- To execute any of the cryptographic commands which are offered by the CryptoServer if run in FIPS-mode, the *CSI-library* can be used. This CSI-library is a C-library which is created for the FIPS-specific cryptographic services and which can be attached by an application on the host PC; it also contains the CSAPI. Only *Cryptographic Users* but no *Administrators* have the right to perform these cryptographic services.

Therefore the CSI-library is not described in this document but in the *Cryptographic User's Guide* [CSFIPS-UserGuide].

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 is announced through the FC and SFC in the header).

2.4.2 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. 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.

2.4.2.1 Authentication State

The CMDS 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.

Example: Authentication state '10000000' means that authentication level 1 is reached in user group 7, whereas there is no authentication for any other user group.

A successful authentication only changes this authentication state. It is up to each individual command, if called, 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 depends on the individual command.

[2] Throughout this chapter, the CryptoServer is considered to be in *operational mode*, see 2.3.4. For a CryptoServer in *boot loader mode* (e. g. if the CryptoServer is in personalization mode and has to be set-up and personalized), other, simplified authentication mechanisms will be applied.

For a CryptoServer in FIPS-mode, the responsible firmware module CMDS will store the authentication state on two different levels:

- **Authentication state of a specific (secure messaging) session:**

  First, CMDS stores a *session-individual authentication state*. This is taken from the authentication of the *GetSession* command and will be stored together with the other session-related data until the session ends (about sessions, see next chapter 2.4.3). The authentication state of the session is only valid within this session, and it gets invalid when the session gets invalid.

- **Authentication state of a specific command:**

  Second, a *command-individual authentication state* will be stored together with the command data. This takes the session-individual authentication state, if the command is performed within a session, and adds the authentication of the respective command (if the command has been authenticated). The command-individual authentication state is only valid for this command.

  This command-individual authentication state is what is checked by the respective command and which is thus crucial for the decision if the command is allowed to be executed or not.

2.4.2.2 Authentication Mechanisms

In FIPS-mode, two different mechanisms for command authentication are implemented:

1. **SHA-1 Hashed Password Authentication:**

   For this mechanism a 16 bytes long operator *password* will be used. First the host demands a 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 which is stored in the user database. Compared with any 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.

2. **RSA Signature Authentication:**

   For this mechanism the host demands again a 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 over the SHA hashed data, compliant to [PKCS#1]). 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.

Both mechanisms are qualified for a secure communication via Ethernet.

For the syntax which has to be used in this context with the CSADM tool, see section 5.9.

2.4.2.3 Users and User Permissions

Commands can only be successfully authenticated by authorized **users**. For the management of these *users*, the CMDS module uses and administrates a **user database** which stores for each user the following data:

- *Name* (which serves as a unique identifier for the user), 8 bytes long.
- *Permissions* of the user. The structure of these user permissions corresponds to the structure of the authentication state, i. e. it consists of eight values each in the range from 0-3, see 2.4.2.1. In FIPS-mode, only some permissions are admissible respectively relevant, see below.
- *Flags* that determines if static login or secure messaging are allowed for this user. In FIPS-mode, these flags are constant:
    - In FIPS-mode, only *single command authentication* is possible. No static login is allowed! Therefore the flag 'no_login' must always be set.
    - *Secure Messaging* is allowed for every user, but the command to open a secure messaging session (*GetSessionKey*) has to be authenticated (see next chapter). Therefore the flag 'sma' must always be set.
- The *authentication mechanism* that has to be used by the user (see above).
- *Authentication data* like RSA key or password, depending on the authentication mechanism, see above.

The CryptoServer provides the appropriate commands to create or delete a user or to change his/her authentication token (data), see section 5.8.

If a user opens and successfully authenticates a Secure Messaging session (*SessionDH* command which has to be authenticated, see next section or 5.10.1), the session-individual authentication state is set to the user's permissions: The permissions of the user are granted to the whole session (i. e. to all commands following the *SessionDH* command).

If a user successfully authenticates any command, the command-individual authentication state is set to the sum of the session-individual authentication state plus the user's permissions. (This includes that, in case the command is *not* performed within a Secure Messaging session, the command-individual authentication state is set to the user's permissions.)

Example:

(Session-individual) authentication state before user's authentication:	01000000
Permissions of the user:	01000001
(Command-individual) authentication state after user's authentication:	02000001

Several users can authenticate one after the other or within one command. Doing this, mixed authentication mechanisms are allowed. All information about the authentication and session is lost if the module is power-cycled.

In FIPS-mode, there are two user groups predefined for the access to the external commands of the standard firmware modules:

1) Users who are allowed to assume the ***Administrator*** role.

   These users must have the *user permission* '22000000' (or higher). All commands for CryptoServer administration and user management (like *LoadFile*, *AddUser*, *SetTime* ...) can only be performed after an *Administrator's* authentication, i. e. when authentication state '22000000' (or higher) has been reached.

2) Users who are allowed to assume the ***Cryptographic User*** role.

   These users must have the *user permission* '00000020' (or higher). All external functions realized by the firmware module CSI which offer cryptographic services (like encryption/decryption, MAC calculation, hashing, ...) and key management functions (like key generation, key import/export, ...) can only be performed after a *Cryptographic User's* authentication, i. e. when authentication state '00000020' (or higher) has been reached.

3) Additionally it is possible to create users which can assume both *Administrator* and *Cryptographic User* role, i. e. with user permission '22000020' (or higher).

If users with other permissions are created, these additional permissions will be of no use in FIPS-mode: Permissions in user groups other than 7, 6 and 1 (e. g. a user permission like '00100000') do not have any influence on the possibilities to authenticate any of the external commands that are given in FIPS-mode.

Upon correct authentication, the (command-individual or session-individual) authentication state will be augmented by the permissions of the user and thus the role (*Administrator* or *Cryptographic User*) is selected based on the user name of the operator.

A user with user name ADMIN who is authorized to assume the Administrator *role is always present in the CryptoServer.*

One first user with user name ADMIN and with the user permission '22000000' to assume the *Administrator* role is always present in the CryptoServer. The authentication mechanism of this predefined user 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.

2.4.3 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 encrypted and integrity-protected with a Triple-DES MAC. For this purpose a secure messaging header data block can be added to the command and answer data block.

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 a Triple-DES session key. The session key will be negotiated with the *Diffie-Hellman* key establishment protocol (see [PKCS#3]).

   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.

2. **Send encrypted commands.**

   The host can send commands to the CryptoServer that are encrypted and integrity protected with a Triple-DES MAC using the previously established session key and the secure messaging layer (software layer SM, which inside the CryptoServer is processed by firmware module CMDS). 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 DES 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 and MAC-protected 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.*
- *Up to four sessions can be opened simultaneously. If a host application requests a new session key while the maximum of 4 sessions are already active, the oldest session is closed and its session key is invalidated.*
- *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 SessionDH *command (see 5.10.1), together in one command line with the command(s) for which secure messaging should be used.*

If the *SessionDH* command is called over the CSADM tool, CSADM will automatically open the session (by getting the session key), perform the specific given 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 5.10.

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. Throughout its life cycle, a CryptoServer will run through the *global states* described in more detail in the following chapters.

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 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. It is located within the manufacturer's secure environment.** The responsible person or organization is able to load the public key $\mathbf{K_{PROD\text{-}PUB}}$ which is related to the manufacturer.

State	Description	Responsibility
produced	The boot loader is running and the manufacturer's public *Production Key* $\mathbf{K_{PROD-PUB}}$ 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* $\mathbf{K_{INIT-PUB}}$ is loaded. Together with $\mathbf{K_{INIT-PUB}}$ the manufacturer's public key for module signature $\mathbf{K_{MDL-SIG-PUB}}$ 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.	
defect	During the boot process, e. g. after a reset, 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.

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 by clearing the CryptoServer (which is only possible if the CryptoServer is either not in FIPS-mode or if it leaves FIPS-mode):

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, if the module is not in FIPS-mode, the user has the possibility to reset the CryptoServer on purpose to one of the previous states: thereby the clearing command *BLClear* (which has to be suitably authenticated, see 3.5 and 5.11.2) can be used. If the CryptoServer is in FIPS-mode, it can be erased to the *initialized* state manually, see 4.5.

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.

The CryptoServer state manufactured *exclusively occurs at the manufacturer's site and never at the customer's site.*

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 it is taken into the *initialized* state, i. e. until the *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$ has been loaded, see next subsection.

The command to load this *Initialization Key* has to be authenticated with the *Production Key*. In the *produced* state, only the manufacturer can perform command authentication because only he is in possession of the private part of the *Production Key*.

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 (which can only be done in the *produced* state and only by the manufacturer who has previously authenticated himself with the *Production Key*), the CryptoServer's global state is set to *initialized*. If the *Initialization Key* is customer-specific, at this point a direct connection is set up for the first time between CryptoServer and client.

The CryptoServer state initialized *also occurs at the customer's site: This can happen if the CryptoServer is in personalization mode (see 2.3.3).*

If the CryptoServer is in FIPS-mode, the global state* initialized *will not occur.

In the *initialized* state, command authentication is additionally possible with the help of a RSA signature of the command which is done by the private part of the *Initialization Key* $\mathbf{K_{INIT\text{-}PRV}}$, and which can be verified by the CryptoServer using the public part $\mathbf{K_{INIT\text{-}PUB}}$. 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.

Together with the *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$ 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 $\mathbf{K_{CS2}}$ (24 bytes DES key) 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.

At the customer's site, the global state *initialized* can occur when the CryptoServer is in personalization mode (i. e. not in FIPS mode!) and the boot loader is active (i. e. the CryptoServer is in the so-called *boot loader mode*, see 2.3.4). For the commands that will be accepted by the CryptoServer in the global state *initialized*, see section 5.11. In particular there are commands available to clear the CryptoServer and to load new firmware modules.

Even if the operating system and further firmware modules are already loaded, but if the CryptoServer is not yet in FIPS mode, 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 is in *boot loader mode*), 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 (what implies that the CryptoServer is not in FIPS-mode but in personalization mode), the customer is able to load the operating system module SMOS and the basic firmware modules CMDS, UTIL and ADM (see 2.3.2) containing the base functionality for download and communication. This service is offered in personalization mode by the *BLLoadFile* boot loader command which has to be signed with the private part $\mathbf{K_{INIT\text{-}PRV}}$ of the customer's *Initialization Key*.[3] 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. In particular the CryptoServer can only enter FIPS-mode (and offer the respective command interface) if the respective FIPS firmware is 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".*

In the *operational* state the boot loader is no longer active. 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.

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. In FIPS mode, this can only happen if the CryptoServer is in FIPS error state.*
For a CryptoServer in FIPS mode which shall execute any cryptographic or other security relevant command, it is mandatory to be in operational *state.*

[3] 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 enter the *defect* state. This test is always run at the beginning of the boot phase after the zeroization 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.

Status information can be retrieved over the GetState *command.*

The defect *state can occur both in FIPS-mode and in personalization mode. If a CryptoServer in FIPS mode is in the* defect *state, this implies that it is in FIPS error state.*
But since for a CryptoServer in defect *state the file system is not yet initialized, it cannot decide if it is in FIPS-mode or not (which should be decided according to the presence of the firmware module FIPS140 in the file system). Thus a CryptoServer in* defect *state will never announce (in the answer to the* GetState *command) that it is in FIPS mode (and FIPS error state), even if this will be the case.*

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

3.3 Alarm

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* is given if the boot loader detects an alarm condition in the alarm status register during the boot process. All firmware and data will be deleted and the CryptoServer will leave FIPS mode. The CryptoServer responds with the current alarm condition and, before any other command execution, a *BLResetAlarm* command is required to reset the alarm.

After an alarm, the CryptoServer will enter personalization mode.
To enter FIPS mode again, the whole process to set-up and personalize the CryptoServer has to be performed, see 6.2 and 4.4.

A more detailed description of the alarm mechanism follows now.

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 4 milliseconds 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 and in personalization mode.*

After an alarm, the CryptoServer contains only the boot loader code and no further firmware. Furthermore, the manufacturer's public *Production Key* $\mathbf{K_{PROD\text{-}PUB}}$ and the customer's public *Initialization Key* $\mathbf{K_{INIT\text{-}PUB}}$ (together with the manufacturer's public key for firmware signature $\mathbf{K_{MDL\ SIG\ PUB}}$) 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. [4]

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 or state 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 meantime, 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 (see 6.2 and 5.11.6).*
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.

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

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 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 6.2.

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.

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*.

The only way to get the CryptoServer out of the power down mode is to reset or power-cycle the module.

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 Command

In personalization mode the CryptoServer offers a clear command (*BLClear*, see 5.11.2). This clear command, which is realized by the boot loader firmware module, is required to give the customer the possibility to erase the CryptoServer without initiating an alarm condition. It is only available in personalization mode, to be used during the process of set-up and personalization of the CryptoServer before FIPS-mode is entered.

The BLClear *command is not available in FIPS-mode!*
It can only be performed in personalization mode. With the BLClear *command the CryptoServer can be cleared to the* initialized *state, where only the boot loader firmware is stored. This command has to be signed with the customer's* Initialization Key $\mathbf{K_{INIT\text{-}PRV}}$.

The *BLClear* command will erase

- CryptoServer's local master key $\mathbf{K_{CS2}}$ (by generating and storing a new $\mathbf{K_{CS2}}$) 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 $\mathbf{K_{PROD\text{-}PUB}}$, $\mathbf{K_{INIT\text{-}PUB}}$ and $\mathbf{K_{MDL\text{-}SIG\text{-}PUB}}$, 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 (and in personalization mode).

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 public part of the key **$K_{PROD-PUB}$** will be imported into the CryptoServer by the manufacturer during the production process. It will be stored in the CryptoServer's flash file. Only the manufacturer has the possibility to clear/change the key (this is only possible if the module has left FIPS mode and entered personalization mode).
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 command to load the Initialization Key has to be signed with the **$K_{PROD-PRV}$**.

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. In personalization mode, **$K_{INIT-PUB}$** can be changed by the customer via the *BLChangeInitKey* boot loader command, see 5.11.3. 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). In FIPS mode, **$K_{INIT-PUB}$** cannot be changed.
usage:	The Initialization Key is needed to prevent unauthorized access after CryptoServer's initialization (e. g. during shipment). **In personalization mode**, the key must be used to authenticate the security relevant commands that are necessary to set-up and personalize the CryptoServer: *BLLoadFile, BLSetRTC, BLResetAlarm, BLChangeInitKey* and *BLClear* will only be performed by the CryptoServer if they are signed by the **$K_{INIT-PRV}$**. **In FIPS-mode**, the owner of the Initialization Key is automatically allowed to assume the role of an *Administrator*. The Initialization Key can thus 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 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 2.4.2. **Attention:** On this basis it is possible to define further users with other authentication mechanisms (*CreateUser* command) who would then also have the permission to assume the *Administrator* role.

	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 calculated, see 3.7. The public key $\mathbf{K_{INIT\text{-}PUB}}$ inside the CryptoServer will be used to verify the MTC signature when the firmware module is downloaded, and every time the loaded module is started (e. g. after a reset): if its MTC signature cannot be verified, a firmware module will not be loaded respectively not be started.
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.

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 thus leave FIPS-mode) 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 public part of the key **$K_{MDL-SIG-PUB}$** will be imported into the CryptoServer by the manufacturer during the production process. From this point on (state at least *initialized*) the key **$K_{MDL-SIG-PUB}$** is resident inside the CryptoServer's flash file. Only the manufacturer has the possibility to clear/change the key (this is only possible if the module has left FIPS mode and entered personalization mode). 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. It will be used to verify the MMC signature when the firmware module is downloaded, and every time the loaded module is started (e. g. after a reset): if its MMC signature cannot be verified, a firmware module will not be loaded respectively 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:	This key is a 24 bytes triple DES key.
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* 2.1.6.6). If the CryptoServer is in FIPS mode, the **K_{CS2}** cannot be deleted or changed. The life cycle of this key ends up when an alarm occurs to the CryptoServer (in this case the CryptoServer leaves FIPS mode and enters personalization mode). 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). In personalization mode the key will also be erased if the clear command *BLClear* is performed; in this case a new **K_{CS2}** will be immediately generated and overwrite the old one.
usage:	The local master key **K_{CS2}** is used to encrypt sensitive data inside the CryptoServer like secret or private cryptographic keys and authentication data for their internal storage.[5] For this purpose it will be used internally by other firmware modules. **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}***

[5] 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.

3.7 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 has to be transported 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*.

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 / Utimaco). The MMC signature has to be 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).

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 has to be done with the private part of the customer's Initialization Key $\mathbf{K_{INIT\text{-}PRV}}$. Its purpose is to give the CryptoServer (who stores the public opponent $\mathbf{K_{INIT\text{-}PUB}}$ of the Initialization Key, see 3.6.2) the possibility to check whether a downloaded module is authentic from the customer the CryptoServer belongs to, and to check its integrity.

Both signatures (MMC and MTC signature) are mandatory: they will be checked both when the firmware module is loaded and every time the CryptoServer is (re)started. If one of the signatures cannot be verified, the firmware module will not be loaded/started.

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/Utimaco)</td><td></td></tr>
<tr><td rowspan="2">MTC Header</td><td rowspan="2">MMC Header</td><td>Binary
(*.out, from Utimaco)</td><td>MMC Signature
(done with Module Signature Key)</td><td>MTC-Signature
(done with Initialization Key)</td></tr>
<tr><td colspan="4">Module Transport Container – MTC
(*.mtc, generated by customer)</td></tr>
</table>

3.7.2 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

The firmware module delivered by the manufacturer/developer will be in MMC format (signed with the module signature key $\mathbf{K_{MDL\text{-}SIG\text{-}PRV}}$).

To prepare the firmware module for the download onto his CryptoServer, the customer must envelope the '*.mmc' files respectively into a MTC using the corresponding *MakeMTC* command of the Administration Tool CSADM, see 5.5.1. The MTC signature has to be done by the customer himself with the private part $\mathbf{K_{INIT\text{-}PRV}}$ of his Initialization Key (see 3.6.2).

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).

4 Typical Administration Tasks

In this chapter the most important administration tasks are explained step by step.

Security-relevant administration tasks can only be performed by a user who is at least allowed to assume the Administrator *role.*
Most of these administration tasks can only be performed by the CryptoServer's System Administrator ADMIN*, i. e. the operator who is allowed to use the CryptoServer's* Initialization Key*.*

4.1 How to Install the CryptoServer

For the hardware installation of the CryptoServer PCI plug-in card on the host PC, and for physical security advices, see [CSInstall-Manual].

To ensure the operational safety, please read the installation manual carefully before unpacking and installing the CryptoServer. Keep the manual always to hand.

Furthermore technical data and instructions for the following situations can be found in this installation manual:

- installation of the host software,
- battery replacement,
- de-installation,
- transport and
- storage.

4.2 How to Get State Indicators

For all administrator's tasks it is vital to be informed about the CryptoServer's actual state and mode.

Precondition:

None.

What to do:

Perform a *GetState* command (see 5.7.1).

CryptoServer's state and mode can be retrieved by analyzing the command output:

CryptoServer's state/mode	Required Indicator	Remarks
CryptoServer is in *FIPS-mode*.	*GetState* command returns `FIPS mode = ON`	
CryptoServer is in *personalization mode*.	Line '`FIPS mode = ON`' is missing in answer of *GetState* command	A CryptoServer is defined to be in personalization mode if it is (not in power-down mode and) not in FIPS-mode, see 2.3.4.

CryptoServer is in *Boot Loader Error State*.	*GetState* command returns
	`state = INITIALIZED (0x00040004)`
	`FIPS mode = ON`
	`FIPS error state (0xb0070039)`
	`temp = …`
	or
	`state = DEFECT (0x00000001)`
	`temp = …`
		The number behind the '`FIPS error state`' indicator serves for error analysis and serves here just as an example.
		(If on the other hand a CryptoServer in FIPS mode is *not* in any error state, the line '`FIPS error state`' is completely left out.)
		The only possibility for a CryptoServer in FIPS mode to be in *initialized* or *defect* state is when it is in FIPS error state.
		A CryptoServer in *defect* state has found a defect boot loader code or file system and can thus not decide yet if it is in FIPS-mode or not.

CryptoServer's state/mode	Required Indicator	Remarks

CryptoServer is in *OS Error State*.	*GetState* command returns
	`state = OPERATIONAL (0x00040005)`
	`FIPS mode = ON`
	`FIPS error state (0xb0830025)`
		CryptoServer is in FIPS mode, but a FIPS error has been noticed when the operating system was already up and running.
		The number behind the '`FIPS error state`' indicator serves for error analysis and serves here just as an example.
		(If on the other hand a CryptoServer in FIPS mode is *not* in any error state, the line '`FIPS error state`' is completely left out.)

CryptoServer is in alarm state.	*GetState* command returns `alarm = ON`	If alarm is given, the CryptoServer is automatically in *initialized* state and in personalization mode, i. e. it has left FIPS mode. (The alarm has cleared all data and firmware modules.) The detailed alarm reasons can be seen from the '`sens = (…)`' text which follows the '`alarm = ON`' line. See example below. See 6.2 for alarm treatment.
No alarm is given.	*GetState* command returns `alarm = OFF`	

CryptoServer's state/mode	Required Indicator	Remarks
CryptoServer is in *dead* state or power down mode.	CryptoServer does not answer to *GetState* command.	Here either the CryptoServer's Central Processing Unit (CPU) is set into power save mode (dead state) or the whole module is without power. The module is therefore unable to perform any action or service. To leave this state, reset or power-cycle the CryptoServer. (One reason for power down mode could be that the CryptoServer's internal temperature has exceeded its operational temperature range from +5°C to +58°C. In this case the module has to be cooled down before a successful reset.)

The state indicators can occur in various (but not all) combinations (see also 2.3.4.2). For a better understanding, here for every mode/state combinations that can occur at the customer's site, one example of the output of the *GetState* command is given. Variations can only occur in the specific error reasons or alarm reasons.

Examples:

***GetState* command answers with …**	**Explanation**

`mode = Operational Mode`
`state = OPERATIONAL (0x00040005)`
`FIPS mode = ON`
`temp = 34,5 [C]`
`alarm = OFF`
`bl_ver = (…)`
`(…)`
	CryptoServer is in **FIPS mode** and **normal operational state** (i. e. no FIPS error has occurred):
	All administrative and cryptographic services are available.

`mode = Bootloader Mode`
`state = INITIALIZED (0x00000004)`
`temp = 35,5 [C]`
`alarm = OFF`
`bl_ver = (…)`
`(…)`
	CryptoServer is in **personalization mode** (i. e. not in FIPS mode!) and in **bootloader mode** (i. e. the bootloader is active).
	The CryptoServer is ready for the personalization process, see 4.4. This process can only be performed by the CryptoServer's *System Administrator*.

***GetState* command answers with …**	**Explanation**

`mode = Operational Mode`
`state = OPERATIONAL (0x00000005)`
`temp = 35,0 [C]`
`alarm = OFF`
`bl_ver = (…)`
`(…)`
	CryptoServer is in **personalization mode** (i. e. not in FIPS mode!) and in **operational mode** (i. e. the operating system SMOS is already loaded and active).
	This state and mode usually occurs in the second phase of the personalization process, see 4.4: steps 8-11 will be performed in this state. This personalization process can only be performed by an *Administrator*.

`mode = Bootloader Mode`
`state = INITIALIZED (0x00040004)`
`FIPS mode = ON`
`FIPS error state (0xb0070039)`
`temp = 34,5 [C]`
`alarm = OFF`
`bl_ver = (…)`
`(…)`
	CryptoServer is in **FIPS mode**, but in **Boot Loader error state** (i. e. a FIPS error has been noticed during the first phase of the CryptoServer's boot process).
	For error analysis, the error number behind `FIPS error state` indicates which specific FIPS error has been occurred.
	Only basic status request services are available. For leaving the FIPS error state, see 4.3.

`mode = Operational Mode`
`state = OPERATIONAL (0x00040005)`
`FIPS mode = ON`
`FIPS error state (0xb0830025)`
`temp = 34,5 [C]`
`alarm = OFF`
`bl_ver = (…)`
`(…)`
	CryptoServer is in **FIPS mode**, but in **OS error state** (i. e. a FIPS error has been noticed when the operating system was already up; in particular the OS is still active; the CryptoServer is thus necessarily in operational state and operational mode).
	For error analysis, the error number behind `FIPS error state` indicates which specific FIPS error has been occurred.
	Only non-sensitive services that have not to be authenticated (like status request services) are available. For leaving the FIPS error state, see 4.3.

***GetState* command answers with …**	**Explanation**

`mode = Bootloader Mode`
`state = INITIALIZED (0x00027f84)`
`temp = 35,0 [C]`
`alarm = ON`
`sens = 027f`
`- Alarm has occurred`
`- external Erase is executed`
 	
`bl_ver = (…)`
`(…)`
	CryptoServer is in **personalization mode** (i. e. not in FIPS mode!) since an **alarm has occurred**. (As always after an alarm has occured, the module is in initialized state and bootloader mode: The alarm has cleared all data and firmware modules.)
	
	No command can be performed before the alarm is set back. See section 6.2 for alarm treatment.
	
	The (hexadecimal coded) two bytes behind '`sens =`' display the contents of the sensory register. To help any user to analyze the alarm reasons, the following text explains these contents:
	
	• '`Alarm has occurred`' means that the physical alarm reason is no longer present. (Alternatively, '`Alarm is present`' would indicate that the physical alarm reason is still present.)
	
	• The individual alarm reason is specified in the following line (here: '`external Erase is executed`').
	
	For more information about possible alarms, see section 3.3.

`mode = Bootloader Mode`
`state = DEFECT (0x00000001)`
`temp = …`
	The reason for a CryptoServer to be in *defect* state is either defect boot loader code or a defect file system. Therefore the module itself cannot decide if it is in FIPS-mode or not.
	
	If the CryptoServer has been in FIPS mode before, the *defect* state is considered to be a FIPS error state.
	
	The customer is not able to get a *defect* CryptoServer working again. Thus the manufacturer/Utimaco has to be contacted.

4.3 How to Quit an Error State

In this chapter it will be explained how to leave a FIPS error state (*Boot Loader Error State* or *OS Error State*). Depending on the error cause this can require various activities.

Precondition:

The CryptoServer is (not in *defect* state but) in any FIPS error state, i. e. the *GetState* command (see chapter 5.7.1) answers with '**FIPS error state = ON**'.[6]

What to do:

1. Perform the *Reset* command or power-cycle the CryptoServer.
2. Check if the module is still in FIPS error state by performing the *GetState* command. If no, you are ready. If yes, go to step 3.
3. Remove the power from the CryptoServer for at least 30 seconds. Power on the CryptoServer again.
4. Check if the module is still in FIPS error state by performing the *GetState* command. If no, you are ready. If yes, go to step 5.
5. The CryptoServer has to be erased completely. To do this, trigger artificially an alarm by performing an *External Erase*: This has to be done manually by a short-circuit of the 'External Erase' pins on the PCI-card (see picture on page 69).
6. Execute *GetState* again to check the success. The alarm must be indicated, but the alarm reason is no longer present, i. e. '**alarm = ON**' and '**alarm has occured**' should be returned.[7]
7. If this is not the case: please contact the manufacturer/Utimaco.
8. Else: The CryptoServer is in personalization mode now. Perform the *BLResetAlarm* command to reset the alarm (see chapter 5.11.6).
9. Execute *GetState* again. The alarm state should now be 'OFF' ('**alarm = OFF**'). If this is not the case, please contact the manufacturer/Utimaco.
10. Else: Since the alarm has cleared all data and firmware modules, the CryptoServer is in personalization mode and a complete new personalization of the CryptoServer must be performed now (see chapter 4.4). If this was successful, the CryptoServer has entered FIPS mode again.
11. Execute *GetState* to check if the CryptoServer is still in FIPS error state. If this is not the case, you are ready. If the module is still in any FIPS error state, please contact the manufacturer/Utimaco.

[6] If the CryptoServer is in *defect* state (i. e. the *GetState* command returns '**state = defect**'), please contact the manufacturer/Utimaco.

[7] In the following line the alarm reason '**external Erase is executed**' will be displayed.

4.4 How to Enter FIPS-Mode: Setup and First Personalization of a CryptoServer

If you receive a new CryptoServer from Utimaco, the module will be in personalization mode and no firmware modules are loaded yet. Thus prior to start operating the CryptoServer in FIPS-mode, a personalization process has to be performed. This personalization process can furthermore be necessary

- after a physical alarm has been occurred to the CryptoServer,
- after the CryptoServer has been cleared on purpose (e. g. to quit certain FIPS error states).

In this chapter it will be explained step by step how a CryptoServer can be personalized in order to enter FIPS-mode.

The process of personalizing a CryptoServer can only be performed by the CryptoServer's System Administrator*, i. e. by an Administrator who is allowed to use the CryptoServer's* Initialization Key *as authentication token.*
Other Administrators are not able to setup and personalize the CryptoServer!

Precondition:

The CryptoServer must be in personalization mode and in *initialized* state. If this is not the case, see section 4.5.

Furthermore all firmware modules that are mandatory in FIPS mode are needed; see chapter 7 for a complete list. The files must be in MTC format (files '*.mtc') and properly signed with the customer specific *Initialization Key* (see section 4.7 how to sign firmware modules).

Furthermore the smart card with CryptoServer's *Initialization Key* is available.

What to do:

1. In order to be able to administrate the CryptoServer, it is a basic requirement that your customer-specific *Initialization Key* is loaded. This is usually the case after CryptoServer's shipment. If the CryptoServer has still loaded Utimaco's standard *Initialization Key*, as first action the *Initialization Key* has to be changed. See 4.8 for a detailed description.
2. Now your customer-specific *Initialization Key* is loaded. Only the customer is able to administrate the CryptoServer (and to download firmware).
   Perform a *GetState* command. The CryptoServer should be in *initialized* state and the alarm state should be 'off'.
3. For further personalization of the CryptoServer you need the private part of the customer specific *Initialization Key* (which is usually on the smart card).
4. Additionally you need all in chapter 7 listed firmware modules as '*.mtc' files, properly signed with the customer specific *Initialization Key* (see section 4.7 how to sign firmware modules).
5. Optional (if the CryptoServer is set up the first time): Set the CryptoServer's clock with the *BLSetRTC* command (see 5.11.5).
6. Load the base firmware modules (SMOS, CMDS, ADM and UTIL, as MTC files) using the *BLLoadFile* command (see 5.11.4).
7. Start the base firmware modules with the *StartOS* command (see 5.11.10).
8. Wait a few seconds, then verify if the CryptoServer is now in *operational* state using the *GetState* command (see 5.7.1).
9. Load all other firmware modules that are listed above with the *LoadFile* command (see 5.7.3).
10. Issue a *Restart* command to activate all new firmware (see 5.6.3). Herewith the CryptoServer goes to the *operational* state.
11. Verify whether the CryptoServer is now in FIPS mode and is not in any error state (using the *GetState* command). If this is not the case: you can check if all firmware modules have been started properly using the *ListModulesActive* command (see 5.7.7). If some firmware modules have not been initialized yet, use the *GetBootLog* command (see 5.7.8) to analyze the problem.

4.5 How to Enter Personalization Mode and to Clear the CryptoServer

It may be useful respectively necessary to clear all data inside of a CryptoServer for example in the following situations:

- You want to securely clear all secret data inside a CryptoServer.
- You want to change the *Initialization Key*.
- You want to leave FIPS-mode, e. g. in order to load and use other (non-FIPS) software.
- The CryptoServer is not workable, e. g. because of buggy basic firmware modules.

At the end of this clearance process, the CryptoServer will be in personalization mode.

Precondition:

The smart card with CryptoServer's *Initialization Key* is available.

Be aware that all data stored on the CryptoServer will be lost, in particular all cryptographic keys except for the *Initialization Key*.

What to do:

If the CryptoServer is not in FIPS-mode, perform the following (simplier) steps to clear the CryptoServer:

1. If the CryptoServer is not in *initialized* state issue a *ResetToBL* command to switch the CryptoServer into *initialized* state (see 5.6.2).
2. Erase all firmware modules and data using the *BLClear* command (see 5.11.2).

If the CryptoServer is in FIPS mode, the following steps have to be performed:

3. The CryptoServer has to be erased completely. To do this, trigger artificially an alarm by performing an *External Erase*: This has to be done manually by a short-circuit of the 'External Erase' pins on the PCI-card (see picture below).
4. Execute *GetState* again to check the success. The alarm must be indicated, but the alarm reason is no longer present, i. e. '**alarm = ON**' and '**alarm has occured**' should be returned.[8] The CryptoServer should be in *initialized* state.
5. If this is not the case: please contact the manufacturer/Utimaco.
6. Else: The CryptoServer is now in personalization mode and *initialized* state. Perform the *BLResetAlarm* command to reset the alarm (see chapter 5.11.6).
7. Execute *GetState* again. The alarm state should now be 'OFF' ('**alarm = OFF**'). If this is not the case, please contact the manufacturer/Utimaco.

[8] In the following line the alarm reason '**external Erase is executed**' will be displayed.

8. Else: The CryptoServer is in personalization mode now and e. g. ready for new software download. See for instance section 4.4 for the process of (re-)personalizing the CryptoServer.

The following picture shows the “External Erase”-pins on the PCI card of the CryptoServer. These two pins have to be short-circuit if the CryptoServer shall be completely cleared:

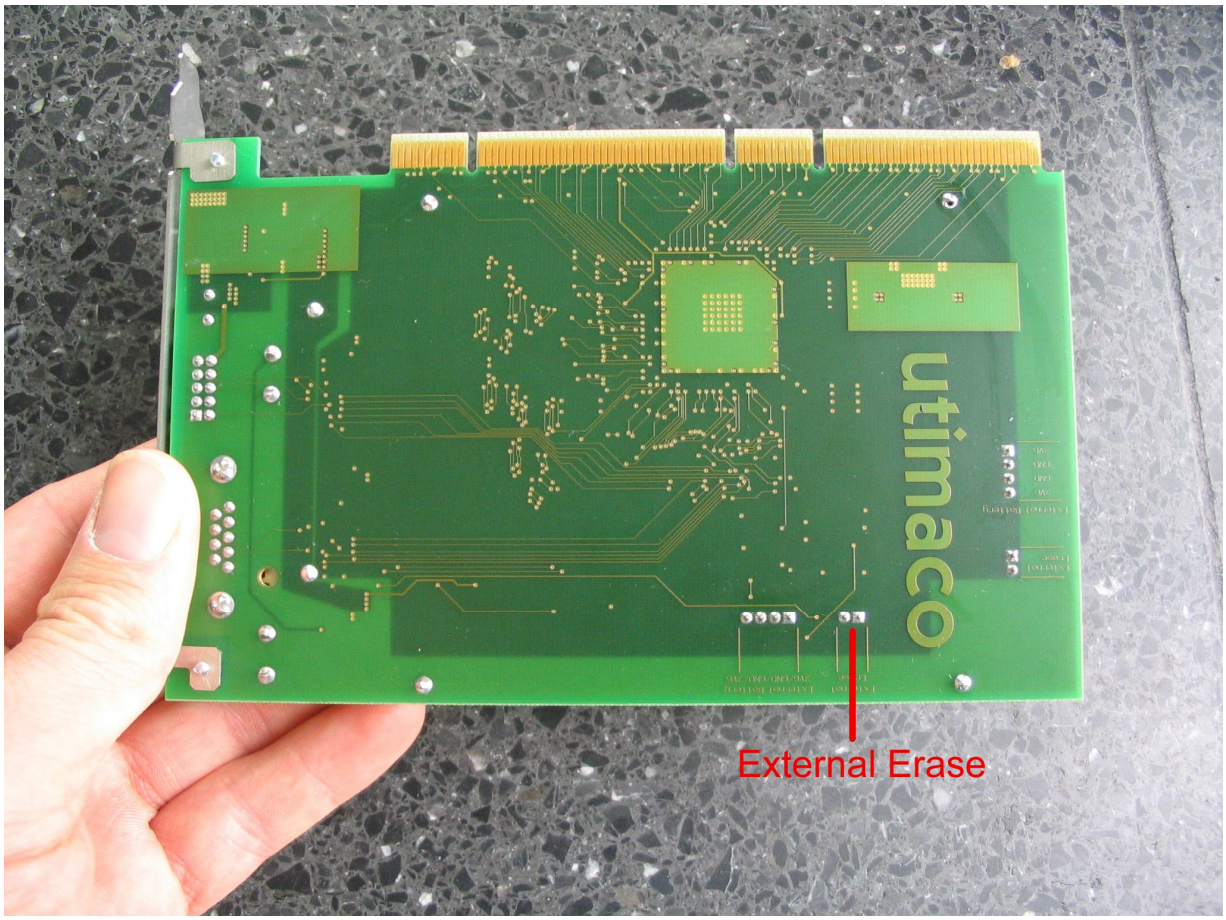

Illustration 4-1: Pins for External Erase

4.6 How to Update Firmware Modules

Precondition:

You have received one or more firmware modules from Utimaco (e. g. containing a bug fix) to be downloaded onto the CryptoServer.

What to do:

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 instead the CryptoServer system uses your customer specific *Initialization Key* you have to (re-)sign the new firmware modules with your *Initialization Key* (refer to section 4.7 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).[9]
5. Download the firmware module(s) ('*.mtc' files) using the *LoadFile* command (see 5.7.3).
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 5.7.7). 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 5.7.8).

You can check if the CryptoServer is still in FIPS mode (and not in any error state) after this process by using the *GetState* command. Be aware that this will only be the case if the appropriate firmware modules in the given versions are loaded (see appendix chapter 7 for a complete list of mandatory firmware modules).

[9] Alternatively also other *Administrators* are allowed to perform the firmware download, in this case their respective authentication token (private RSA key or password) is needed.

4.7 How to (Re-)Sign 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 CryptoServer's *Initialization Key*. Using a customer specific *Initialization Key* on the CryptoServer requires (removing and re-)building the signature of each new firmware module which you receive from Utimaco.

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* (test key).

What to do:

1. Check that the firmware modules have been transferred to the customer in a secure way (e. g. PGP-encrypted and authenticated).
2. In case of (b) the authenticity of the firmware modules can be verified using the *VerifyMTC* command, see 5.5.3
3. In case of (b) the old MTC signature has to be removed from each firmware module with the *RemoveMTC* command, see 5.5.2. 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 applied to the '*.mmc' file, see 5.5.1. The command creates '*.mtc' files which are ready for download onto the CryptoServer.

4.8 How to Change 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.

Keep in mind that this demands a complete clearance of the CryptoServer's data and firmware, in particular all customer-relevant cryptographic keys will be deleted.

Preconditions:

- The CryptoServer is in personalization mode and in *initialized* state. If this is not the case, see 4.5 how to get there.
- 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 smart card for his representative (see 4.9).

What to do:

1. Reset the CryptoServer to boot loader mode using the command *ResetToBL* (see 5.6.2).
2. Execute the *BLChangeInitKey* command (see chapter 5.11.3). 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 *BLClear* command (see 5.11.2).
4. Sign all firmware modules with the new *Initialization Key* (see section 4.7 how to sign firmware modules).
5. In order to enter FIPS-mode again, perform a complete (re-)personalization of the CryptoServer. Refer to section 4.4 for a detailed description of this process.

4.9 How to 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 can under no circumstances 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: Since a back-up smart card contains only one XOR-half of the RSA key, two back-up cards will be 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. Utimaco's KeyTool 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.

5 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). In addition it contains utility functions 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.
- one free serial port to connect the PIN-Pad (smart card reader with keyboard and display).

PC-Software:

- Windows NT (Service Pack 5 or higher) or
- Windows 2000 or
- Linux (any distribution with version of 'libc' equal or higher than 2.2.2)

5.1.1 Installation of the CSADM

The installation of the Administration Tool CSADM on the PC is quite simple.

Installation under Windows:

- Copy the file 'csadm.exe' to a well-chosen directory.
- Add this directory to the 'PATH' environment variable to be able to call the Administration Tool from any other directory.
- 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 5.1.2).

Installation under Linux:

- Copy the executable file 'csadm' to a well-chosen bin directory.
- 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 5.1.2).

5.1.2 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 brackets are optional.*
- *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 PCI:/dev/cryptoserver0 InitPrvKey=d:\keys\cs2\init_dev_prv.key BLLoadFile=c:\firmware\exmp.mtc
- csadm Dev=PCI:1 AuthRSASign=ADMIN,:cs2:cp8:COM2 SetTime=20020602111532

Every command running on CryptoServer requires the 'Dev=' parameter (device parameter, see also 5.4) which sets CryptoServer's address. (Only commands running locally without using a CryptoServer, like module preparation commands, do not need this parameter.) Possible values are:

Address	Description
`PCI:/dev/cryptoserver0`	local CryptoServer No. 1 in a Linux system.
`PCI:0`	local CryptoServer No. 1 in a Windows system.
`PCI:1`	local CryptoServer No. 2 in a Windows system.
`...`	...

If the environment variable CRYPTOSERVER is set according to the above mentioned syntax, the 'Dev=' parameter can be skipped (see also 5.1.1).
Using the 'Dev=' parameter overrides the CRYPTOSERVER environment variable in any case for the specific command.

5.1.3 Key Specifiers

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

(1) RSA keys stored in a file (in plain text)

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

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

- *a filename of a key file or*
- *a smart card specifier.*

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.

5.1.4 Password Entry

To authenticate a security-relevant administration command, an administrator who uses the *SHA-1-Hashed Password Authentication* mechanism has to enter his user name and password to the CSADM tool (see 5.9 and 2.4.2.2). The password will not be transmitted in clear to the CryptoServer but only in a SHA1-hashed form. But 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.

See 5.9.2 for the exact syntax of the SHA-1-hashed password authentication.

For a CryptoServer in FIPS-mode, the usage of hidden password entry is mandatory.

Remember that in FIPS-mode it is mandatory to use passwords of at least 6 characters length!

5.2 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.

The following illustration shows how commands are executed on a local CryptoServer:

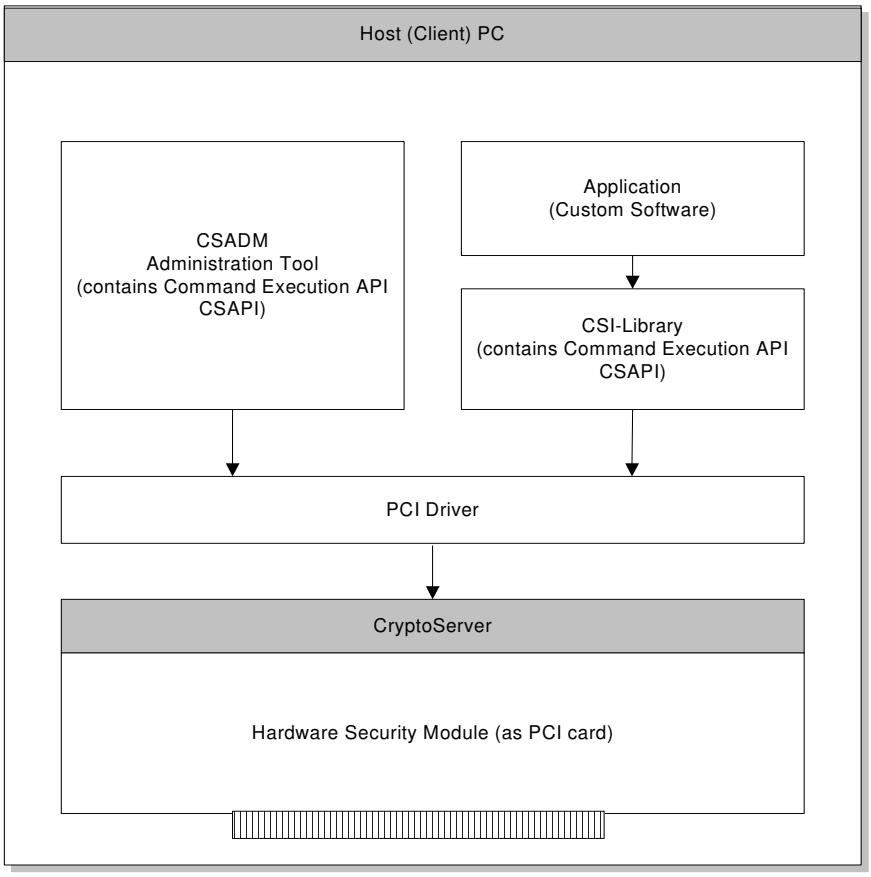

Illustration 5-1: Command Execution

An application on the host PC can use the FIPS-specific library for cryptographic services (CSI-library, which also contains the *command execution library* CSAPI) to execute any of the cryptographic commands which are offered by the CryptoServer if run in FIPS-mode. (*Cryptographic Users* have the right to perform these cryptographic services.)

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).

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

5.3 Basic Commands

These basic commands are CSADM internal functions.

No connection to a CryptoServer will be established.

5.3.1 Help

If called without any parameter, this command shows a list of all available CSADM commands. If the 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]**

5.3.2 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, 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**

5.3.3 Version

This command shows the version of the CSADM.

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

5.4 Commands to Set-Up Parameters

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

The following table shows all available parameters:

Command	Parameter	Used by Command(s)
`Dev=` `Device=`	address of CryptoServer (see 5.1.2)	nearly all
`MTCSignKey=`	key specifier of private key (see 5.1.3).	`MakeMTC`
`MMCSignKey=`	key specifier of private key (see 5.1.3).	`MakeMTC`
`SignType=`	"1" for signature calculation according to PKCS#1v1.5 [PKCS#1] (mandatory in FIPS mode)	`MakeMTC`
`MTCPubKey=`	key specifier of public key (see 5.1.3).	`VerifyMTC`
`MMCPubKey=`	key specifier of public key (see 5.1.3).	`VerifyMTC`
`InitPubKey=`	key specifier of public key (see 5.1.3).	`ChangeInitKey`
`InitPrvKey=`	key specifier of private key (see 5.1.3).	many boot loader commands
`Admin3=`	string of up to 16 characters.	`ChangeInitKey`
`Pause=`	"1" or "0", to switch on or off	

If the environment variable CRYPTOSERVER is set, the 'Dev=' parameter can be skipped (see also 5.1.1).
Using the 'Dev=' parameter overrides the CRYPTOSERVER environment variable in any case for this specific command.

5.5 Commands to Prepare Firmware Modules

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

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

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

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

  Signing the *Module Manufacturer Container* is mandatory. The private part of the Utimaco's *Module Signature Key* will have to be used for calculation of the signature. 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 Utimaco as the author of the firmware module s. t. the CryptoServer - and anyone else - is able to proof the authenticity of the firmware module's origin.

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

  Signing this container is also 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.

The structure and signature of both containers of a firmware module can be checked manually (outside the CryptoServer) using the command *VerifyMTC* (see chapter 5.5.3). 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 (Utimaco) 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 5.11.3).*

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

5.5.1 MakeMTC

This command builds the *Module Transport Container* file (*.mtc) from a *Module Manufacturer Container* file (*.mmc). 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.

Syntax	csadm MTCSignKey=#keyspec SignType=1 MakeMTC=#file
Parameters	#keyspec private part of the *Initialization Key* *(see* 5.1.3 *for possible key specifiers)* #file firmware module ('*.mmc')
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 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. ■ Setting the parameter 'SignType=1' means that the MTC signature will be calculated according to PKCS#1v1.5 [PKCS#1]. This is mandatory in FIPS mode since other MTC signatures would not be accepted by the CrytoServer during firmware download. ■ If the MTCSignKey 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 the command has successfully been performed the signed firmware module is stored in the actual directory.

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.

5.5.2 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 given as input file, its header and signature (if present) 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) firmware module as MMC (*.mmc) (should be avoided)
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 actual 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.

5.5.3 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 (only if MTC is given as input file)* *(see* 5.1.3 *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. • If only a MMC is given as input file, the 'MTCPubKey' parameter has to be left out.

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.

5.5.4 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’ or ‘*.dll’)		
Example	csadm ModuleInfo=adm.mtc		
Output	**Module**	**'ADM'**	// module name
	ID	**87**	// module ID (FC)
	Version	**1.0.2.0**	// 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*)

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

5.5.5 RenameToVersion

This command expands the file name of a firmware module by inserting the version information 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’ or ‘*.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`

5.6 CryptoServer Driver Commands

The commands in this section are directed to the PCI driver of the CryptoServer.

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

5.6.1 Reset

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

If the CryptoServer is not in FIPS-mode *(i. e. the CryptoServer is in personalization mode, e. g. during the initial setup phase), 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).*

In this case it is therefore recommended NOT to use the Reset *command but instead*

- *to use* Restart *(see 5.6.3) to restart the CryptoServer as fast as possible (3 – 5 sec) or*
- *to use* ResetToBL *(see 5.6.2) to reset the CryptoServer and to get into boot loader mode.*

If the CryptoServer is in FIPS-mode *this boot loader time window is skipped, the* Restart *and* ResetToBL *commands are not available and the* Reset *command is the only way to restart the CryptoServer.*

Syntax	csadm [Dev=#device] Reset
Parameter	#device device specifier (see 5.1.2)
Output	none on success or error message

5.6.2 ResetToBL

If the CryptoServer is in FIPS-mode, the ResetToBL *command is not available!*

If the CryptoServer is not in FIPS-mode but in personalization mode, this command can be used during the process to set-up and personalize the CryptoServer, see chapter 4.4.

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 (which will only be accepted if the CryptoServer is not in FIPS-mode). 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 5.11.10).*
- *The* RecoverOS *command (see 5.11.11) 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! In particular the CryptoServer is not in FIPS-mode (but in personalization mode) and no cryptographic service can be executed.*

Syntax	csadm [Dev=#device] ResetToBL
Parameter	#device device specifier (see 5.1.2)
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 until the *StartOS* command is sent to the CryptoServer.

5.6.3 Restart

If the CryptoServer is in FIPS-mode, the Restart *command is not available!*

If the CryptoServer is not in FIPS-mode but in personalization mode, this command can be used during the process to set-up and personalize the CryptoServer, see chapter 4.4.

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 (which will only be accepted if the CryptoServer is not in FIPS-mode) is sent to the CryptoServer immediately.

If the CryptoServer is not in FIPS-mode, using the Restart *command is the fastest way to restart the operating system. If it is successful, the CryptoServer is in operational mode afterwards.*
If the Restart command is performed as last step of the personalization process, and in particular if all firmware modules that are mandatory for FIPS-mode are loaded, the CryptoServer will remain in FIPS-mode afterwards.

Syntax	csadm [Dev=#device] Restart
Parameter	#device device specifier (see 5.1.2)
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.

5.6.4 GetInfo

This command retrieves information from the PCI driver of the CryptoServer. It shows the driver version, PCI slot number, interrupt number, battery state, timeout, 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 requires 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 5.1.2)
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 only for diagnostic purposes in error case!

5.7 Commands for CryptoServer's Administration

The commands described in this chapter offer extended administrative services for a CryptoServer in FIPS-mode.

All administrative commands that are security-relevant

- LoadFile
- DeleteFile
- SetTime

can only be performed if the CryptoServer is not in any FIPS error state!

The four commands

- GetState
- GetAlarmLog
- GetTempLog
- GetTimeLog

request status information; they will be executed in normal operational state (i. e. no FIPS error has occurred) as well as in any FIPS error *state (*Boot Loader Error *state in boot loader mode as well as* OS Error *state in operational mode).*
They are responded by the boot loader (if the CryptoServer is in Boot Loader Error *state) respectively the administration module ADM (otherwise) in the same way.*

These commands will even be executed if the CryptoServer is not in FIPS-mode.

Additionally to the four commands listed above, the following commands

- GetTime
- GetBootLog
- ListFiles
- ListModulesActive
- MemInfo
- Test

can be executed in normal operational state (i. e. no FIPS error has occurred) as well as in OS Error *state. (In both cases the CryptoServer is in operational mode.)*

These commands will neither be executed if the CryptoServer is in BL Error *state, nor if it is in personalization mode.*

Some of the commands have to be authenticated (e. g. *LoadFile*), some do not (e. g. *ListFiles*). If a command has to be authenticated, authentication must be done by a user with *Administrator* rights. For a detail description of the possible authentication mechanisms and the user concept see chapter 2.4.2. The commands for user management can be found in chapter 5.8.

Although command authentication in FIPS-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 which have to be authenticated *does not specify each possibility of authentication*. Instead a placeholder (*<Authentication>*) is inserted; 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 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.
*The exact syntax of the various authentication commands (*AuthRSASign *or* AuthSha1Pwd) *can be found in chapter 5.9.*

5.7.1 GetState

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

The GetState *command is responded by the CryptoServer in any mode and state (in FIPS-mode as well as in personalization mode, and in normal operational state as well as in any* FIPS error *state) 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 5.1.2)
Required State	any
Authentication	none
Output	Example for CryptoServer in FIPS-mode and normal operational state: **mode = Operational Mode** **state = OPERATIONAL (0x00040005)** **FIPS mode = ON** **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** Example for CryptoServer in in FIPS-mode and Boot Loader Error state: **mode = Bootloader Mode** **state = INITIALIZED (0x00040004)** **FIPS mode = ON** **FIPS error state (0xb0070039)** **temp = 47,5 C** **alarm = OFF** **bl_ver = 01000500** **(…) = (…)**

See chapter 4.2 for further examples.

The displayed fields have the following meaning:

field	*description*	
mode	*operational mode* or *bootloader mode, see 2.3.4.* This mode has not to be confused with FIPS-mode (respectively personalization mode), these modes can occur independently from each other!	
state	state of the CryptoServer (*defect, manufactured, produced, initialized* or *operational*; see also 3.2) 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.	
FIPS mode	**ON**: The CryptoServer is in FIPS-mode. If the module is not in FIPS-mode but in personalization mode, this line is left out!	
FIPS error state	If this indicator is returned, the CryptoServer is in FIPS error state. The number in brackets behind the '**FIPS error state**' indicator serves for error analysis; here just an example is given. If no FIPS error has occurred (i. e. the module is not in FIPS error state), this line is left out!	
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!
	< 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: ■ module will be reset
	5°C-58°C	normal operation (CryptoServer ready to work)
	> 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: ■ module will be reset
	> 66°C	sensory triggers a temperature alarm -> all user data on the CryptoServer will be cleared!

field	*description*
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). In the first case, '**alarm is present**' will be displayed, in the second case, '**alarm has occurred**' will be displayed.
version of the boot loader	boot loader version has to be 2.0.2.4 in certified FIPS-mode
version of the hardware	hardware version has to be 2.0.2.0 in certified FIPS-mode
UID	8 bytes long, unique Unit Identifier of CryptoServer's processor (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. Usually the customer's name is registered here.
adm3	16 bytes long field which can be freely assigned with every change of the *Initialization Key* in the CryptoServer's personalization phase (*BLChangeInitKey* command, see 5.11.3). This field may be empty or used otherwise, but it is recommended that here the name of the loaded *Initialization Key* is stored (e. g. 'Init-Live01-Key').

5.7.2 ListFiles

This command lists all files stored on the CryptoServer.

The following directories are available on the CryptoServer:

Directory	Component	Size	Description
\SYS	flash device (see also 2.1.6.2)	416 KBytes	**System directory**. The initial administration keys, the boot loader's configuration file, log files and the back-up copies of the basic firmware modules are stored here.
\FLASH	flash device (see also 2.1.6.2)	15,5 MBytes	**Working directory**. The regular set of firmware modules and any other kind of application data (databases, configuration files or log files, ...) is stored here.
\NVRAM	NV-RAM (see 2.1.6.5)	512 KBytes	**Non-volatile directory**. The NV-RAM is not used in FIPS-mode.

Syntax	csadm [Dev=#device] ListFiles[=#pattern]
Parameters	#device device specifier (see 5.1.2) #pattern file name 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* (if in personalization mode) or *operational* (here also in *OS Error* state available)
Authentication	none

Output

SYS\Init.KeyRsaPub	**168**	**-**	
SYS\MdlSg.KeyRsaPub	**168**	**-**	
SYS\Prod.KeyRsaPub	**168**	**-**	
SYS\adm.mtc	**30920**	**ADM**	**0x87 2.0.0.2 Administration Module**
SYS\bl.ini	**72**	**-**	
SYS\cmds.mtc	**49504**	**CMDS**	**0x83 1.0.5.0 Command Scheduler**
SYS\smos.mtc	**92240**	**SMOS**	**0x00 1.0.3.7 SMOS**
SYS\util.mtc	**32088**	**UTIL**	**0x86 1.0.6.0 Utility Module**
8 files	**20532**	**8 bytes**	
FLASH\CERT.db	**4105**	**-**	
FLASH\RSA.db	**18593**	**-**	

	FLASH\adm.mtc	**30920**	**ADM**	**0x87 1.0.2.0 Administration Module**
	FLASH\asn1.mtc	**12088**	**ASN1**	**0x91 1.0.1.0 Asn1 Module**
	FLASH\cmds.mtc	**49504**	**CMDS**	**0x83 1.0.5.0 Command Scheduler**
	FLASH\db.mtc	**37288**	**DB**	**0x88 1.0.1.1 Database module**
	FLASH\hash.mtc	**36136**	**HASH**	**0x89 1.0.1.0 Hash Module**
	FLASH\lna.mtc	**47360**	**LNA**	**0x8e 1.0.3.0 LNA**
	FLASH\rkmg.mtc	**44680**	**RKMG**	**0xd2 0.9.2.1 RSA Keymanagement**
	FLASH\sigega.mtc	**28304**	**SIGEGA**	**0xc6 0.9.1.7 Signature Module**
	FLASH\smos.mtc	**92240**	**SMOS**	**0x00 1.0.3.7 SMOS**
	FLASH\time.log	**14**	**-**	
	FLASH\user.db	**60**	**-**	
	FLASH\util.mtc	**32104**	**UTIL**	**0x86 1.0.6.0 Utility Module**
	FLASH\vdes.mtc	**26944**	**VDES**	**0x81 1.0.0.3 DES Module**
	FLASH\vrsa.mtc	**44648**	**VRSA**	**0x84 1.0.3.3 RSA Module**
	16 files	**504988**	**bytes**	
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) ▣ version number ▣ long name			

5.7.3 LoadFile

This command loads a file (usually firmware modules) onto the working directory (\FLASH) 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.

Syntax	csadm [Dev=#device] <*Authentication*> LoadFile=#file
Parameter	#device device specifier (see 5.1.2) #file file name (eventually with path)
Examples	csadm AuthRSASign=ADMIN,d:\keys\init_prv.key … … LoadFile=exmp_dbg_1.0.2.3.mtc
Required State	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
Output	none on success or error message
Note	■ If the file name 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 actual 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! ■ File names 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.

5.7.4 DeleteFile

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

If a firmware module in the working directory (\FLASH) has been deleted, 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 after the CryptoServer is restarted the module becomes inactive (see Restart *command, chapter 5.6.3).*

Syntax	csadm [Dev=#device] *<Authentication>* DeleteFile=#file
Parameters	#device device specifier (see 5.1.2) #file file name
Examples	csadm AuthSHA1Pwd=paul,swordfish DeleteFile=exmp.mtc
required state	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
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.

5.7.5 GetTime

This command returns the CryptoServer's system time.

Syntax	csadm [Dev=#device] GetTime
Parameter	#device device specifier (see 5.1.2)
required state	*initialized* (if in personalization mode) or *operational* (here also in *OS Error* state available)
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).

5.7.6 SetTime

This command sets the CryptoServer's system clock.

Syntax	csadm [Dev=#device] *<Authentication>* SetTime=#time
Parameters	#device device specifier (see 5.1.2) #time 'YYYYMMDDHHMMSS' or 'SYSTEM'
Examples	■ csadm AuthRSASign=ADMIN,:cm8:usa:COM1 SetTime=SYSTEM ■ csadm AuthSHA1Pwd=paul,swordfish SetTime=20020602115500
required state	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
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. ■ 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 5.7.11) 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

5.7.7 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 5.7.8) provides more detailed information about the reason.*

Syntax	csadm [Dev=#device] ListModulesActive
Parameters	#device device specifier (see 5.1.2)
required state	*operational* (also in *OS Error* state available)
Authentication	none
Output	**0 SMOS 1.0.3.7 OS_MDL_INIT_OK** **1 FIPS140 1.0.0.2 OS_MDL_INIT_OK** **65 CSI 1.0.0.3 OS_MDL_INIT_OK** **81 VDES 1.0.0.3 OS_MDL_INIT_OK** **83 CMDS 1.0.5.0 OS_MDL_INIT_OK** **84 VRSA 1.0.3.3 OS_MDL_INIT_OK** **86 UTIL 1.0.6.0 OS_MDL_INIT_OK** **87 ADM 1.0.2.0 OS_MDL_INIT_OK** **88 DB 1.0.1.1 OS_MDL_INIT_OK** **89 HASH 1.0.1.0 OS_MDL_INIT_OK** **8b AES 1.0.0.0 OS_MDL_INIT_OK** **8e LNA 1.0.3.0 OS_MDL_INIT_OK** **91 ASN1 1.0.1.1 OS_MDL_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 5.6.3) 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 CSI (Cryptographic Services Interface) module requires the VDES (DES functionality) 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.

5.7.8 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. 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.

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).
The terminal software has to be configured as follows:

- *baudrate: 115200*
- *databits: 8*
- *parity: none*
- *stopbits: 1*

Syntax	csadm [Dev=#device] GetBootLog
Parameter	#device device specifier (see 5.1.2)
Required State	*initialized* or *operational* (any mode or state)
Authentication	none
Output	**SMOS Ver. 1.0.3.7 started** **module 0x83 (CMDS) initialized successfully** **module 0x89 (HASH) 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 0x91 (ASN1) initialized successfully** **module CSI can't find module AES (00000000)** **FATAL: module 0x65 (CSI) initialization failed (err = b065fe01)**

5.7.9 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. If no alarm has occurred since CryptoServer's production, the 'alarm.log' file does not exist.

This command is responded by the CryptoServer in any mode and state (FIPS mode or personalization mode; operational state or any FIPS error state).

Syntax	csadm [Dev=#device] GetAlarmLog
Parameter	#device device specifier (see 5.1.2)
Required State	*initialized* or *operational* (any mode or state)
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 occurs in case of an empty battery) See chapter 6.2 '*Alarm Treatment*' how to deal with an alarm.

5.7.10 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 its starting-up process (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 (see *BLClear*, chapter 5.11.2; this command is not available in FIPS-mode). 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.

This command can be performed by the CryptoServer in any mode and state (FIPS mode or personalization mode; operational state or any FIPS error state).

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

Syntax	csadm [Dev=#device] GetTempLog
Parameter	#device device specifier (see 5.1.2)
Required State	*initialized* or *operational* (any mode or state)
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.

5.7.11 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 5.11.2; this command is not available in FIPS-mode) and can also be explicitly deleted by the user (see *DeleteFile* chapter 5.7.4). If the clock was not set since CryptoServer's last initialization, the 'time.log' file does not exist.

This command can be performed by the CryptoServer in any mode and state (FIPS mode or personalization mode; operational state or any FIPS error state).

Syntax	csadm [Dev=#device] GetTimeLog
Parameter	#device device specifier (see 5.1.2)
required state	*initialized* or *operational* (any mode or state)
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.

5.7.12 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 5.1.2) #dir directory on the CryptoServer ('SYS', 'FLASH' or 'NVRAM')
Example	csadm MemInfo=SYS
Required State	*operational* (also in *OS Error* state available)
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.

5.7.13 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 5.1.2) #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	*operational* (also in *OS Error* state available)
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 ;-)

5.8 User Management

In FIPS-mode, security relevant commands have to be authenticated either by an user who has assumed the ***Administrator*** role, or by an user who has assumed the ***Cryptographic User*** role (see chapter 2.4.2 for the details). In dependency on the command the user therefore has to be fitted out with a specific permission.

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 8 characters (A..Z, a..z, 0..9)
Permission	A user's permission consists of 8 different figures (enumerated downward 7 ... 0) with values between 0 and 3. In FIPS-mode the following permissions are possible: 1) Users who shall be allowed to assume the ***Administrator*** **role** must have the **user permission '22000000'** (or higher). All commands for CryptoServer administration and user management (like *LoadFile*, *AddUser*, *SetTime* ..., see this chapter and 5.7) can only be performed after an *Administrator's* authentication. 2) Users who shall be allowed to assume the ***Cryptographic User*** **role** must have the **user permission '00000020'** (or higher). All external functions realized by the firmware module CSI which offer cryptographic services (like encryption/decryption, MAC calculation, hashing, ...) and key management functions (like key generation, key import/export, ...) can only be performed after a *Cryptographic User's* authentication. 3) Additionally it is possible to create users which can assume both *Administrator* and *Cryptographic User* role, i. e. users with user permission '22000020' (or higher).
Mechanism	■ RSA signature (either with a smart card or a key file, communication runs over the host) ■ SHA-1 hashed password See 2.4.2 for more details about the mechanisms and their advantages / disadvantages.
Flags	■ no_login: user has to authenticate each (sensitive) command separately ■ sma: user may open a secure messaging session if he/she authenticates the command (see 5.10). In FIPS-mode, both flags are mandatory and have to be set.

Since the creation of new users also requires the authentication of a user with Administrator rights, one initial user 'ADMIN' with Administrator's permission '22000000' is

always available. ADMIN uses the 'RSA-Signature' authentication mechanism with the *Initialization Key*, ADMIN is therefore intended to be the System Administrator. The user ADMIN is always present (even if the database 'user.db' has not yet been created) and cannot be deleted.

The initial user 'ADMIN' is always present and has the permission to assume the Administrator *role (in FIPS-mode) respectively to perform the necessary administrative tasks during the process of personalizing the CryptoServer (before FIPS-mode is entered, see 5.11).*
ADMIN 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'.

If in the syntax of one of the commands described in the following chapters the placeholder <Authentication> is found, it has to be replaced by one or more authentication commands according to the required authentication and depending on the specific user's authentication mechanism.
*The exact syntax of the various authentication commands (*AuthRSASign *or* AuthSha1Pwd*) can be found in chapter 5.9*

The commands described in the following subsections are only available if the CryptoServer is in normal operational FIPS-mode and not in any FIPS Error state.

5.8.1 ListUser

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

Syntax	csadm [Dev=#device] ListUser
Parameter	#device device specifier (see 5.1.2)
Required State	*operational,* not available in any FIPS error state
Authentication	none
Output	**Name** **Permission** **Mechanism** **Flags** **ADMIN** **22000000** **RSA sign** **no_login + sma** **paula** **22000000** **sha-1 passwd** **no_login + sma** **paul** **00000020** **sha-1 passwd** **no_login + sma** **test** **22000020** **sha-1 passwd** **no_login + sma**
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*) see the previous pages or 2.4.2.

5.8.2 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 (SHA-1 hash and 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.

Syntax	csadm [Dev=#device] <*Authentication*> … ... AddUserRSASign=#user,#permission,#flags,#keyspec
Parameters	#device device specifier (see 5.1.2) #user user name, up to 8 characters (no restrictions) #permission 8 digits 'XXXXXXXX': • '22000000' for a user with *Administrator* rights, • '00000020' for a user with *Cryptographic User* rights, • '22000020' for a user with Administrator *and* Cryptographic User rights #flags 'no_login+sma' #keyspec public part of the user's RSA key (see 5.1.3)
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 … … AddUserRSASign=Paul,22000000,no_login+sma,:cs2:cp8:COM1
Required State	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
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.

5.8.3 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 5.1.2) #user existing user name #keyspec public part of the new user's RSA key (see 5.1.3)
Example	csadm AuthRSASign=paul,:cs2:cp8:COM1 ChangeUserRSASign=paul,:cs2:cp8:COM1
Required State	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism 'RSA Signature' (see chapters 2.4.2 and 5.9).
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 (user's old RSA key) after that.

5.8.4 AddUserSHA1Pwd

With this command a new user is added to the user database using the 'SHA1 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 5.10), the password will be transmitted in clear to the CryptoServer. Therefore the usage of Secure Messaging is mandatory.*

The CryptoServer can check later the command authentication via a SHA-1 hash value which is going to be added to each command: This hash value will be 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.

This mechanism has the following advantages (compared with any clear password authentication):

- *If a command is authenticated with the 'SHA1 hashed Password' mechanism, 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,#flags,#password	
Parameters	#device	device specifier (see 5.1.2)
	#user	user name, up to 8 characters (no restrictions)
	#permission	8 digits 'XXXXXXXX': • '22000000' for a user with *Administrator* rights, • '00000020' for a user with *Cryptographic User* rights, • '22000020' for a user with Administrator *and* Cryptographic User rights
	#flags	'no_login+sma'
	# password	for hidden password entry: string 'ask' (see 5.1.4 and below; hidden password entry is mandatory in FIPS-mode); otherwise: user password (length between 6 and 16 characters)

Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 AddUserSHA1Pwd=paul,00000020,no_login+sma,ask
required state	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
Output	none on success or error message

In FIPS-mode it is mandatory to use passwords of at least 6 characters length.

For a CryptoServer in FIPS-mode, the usage of hidden password entry is mandatory.

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.

5.8.5 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 5.10), the new password will be transmitted in clear to the CryptoServer. Therefore the usage of secure messaging is mandatory.*

Syntax	csadm [Dev=#device] *<Authentication>* … ... ChangeUserSHA1Pwd=#user,#password
Parameter	#device — device specifier (see 5.1.2) #user — existing user name #password — for hidden password entry: string 'ask' (see 5.1.4 and below; hidden password entry is mandatory in FIPS-mode); otherwise: new user's password (length between 6 and 16 characters)
Example	csadm AuthSHA1Pwd=paul,ask ChangeUserSHA1Pwd=paul,ask
required state	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by the existing user according to his/her authentication mechanism 'SHA-1 Hashed Password' (see chapters 2.4.2 and 5.9).
Output	none on success or error message
Note	The user has to be already present in the user database

In FIPS-mode it is mandatory to use passwords of at least 6 characters length.

For a CryptoServer in FIPS-mode, the usage of hidden password entry is mandatory.

If hidden password entry is used, the CSADM will ask for the new user's 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.

5.8.6 DeleteUser

This function deletes a user from the user database.

Syntax	csadm [Dev=#device] *<Authentication>* DeleteUser=#user
Parameter	#device device specifier (see 5.1.2) #user existing user
Example	csadm AuthRSASign=ADMIN,:cs2:cp8:COM1 DeleteUser=paul
Required State	*operational,* not available in any FIPS error state
Authentication	The command must be authenticated by an *Administrator* (see chapters 2.4.2 and 5.9).
Output	none on success or error message

5.9 Command Authentication

Apart from status requests and other non-security relevant commands all CryptoServer commands require command authentication. Herefore, the CryptoServer offers two different authentication mechanisms. See chapter 2.4.2 for a detailed explanation of the authentication concept.

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 placeholders *<Authentication>* which are given in the syntax of all security relevant commands described throughout this chapter 5.

5.9.1 AuthRSASign

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

The following steps are performed during command execution:

1. A challenge value is requested from the CryptoServer.
2. Command data and challenge value are signed (according to PKCS#1) using the private part of the user's RSA 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 RSA 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] AuthRSASign=#user,#keyspec #command
Parameters	#device device specifier (see 5.1.2) #user user name #keyspec key specifier for private part of the user's RSA key (smart card or key file, see 5.1.3) #command command which has to be authenticated
Examples	csadm AuthRSASign=paul,:cs2:cp8:COM1 DeleteFile=xxx.mtc
required state	*operational,* not available in any FIPS error state
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.

5.9.2 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:

1. A challenge value is requested from the CryptoServer.
2. A SHA-1 hash value is calculated over the user's password, the command data and the challenge value.
3. Command data and hash value are sent to the CryptoServer.
4. 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.
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] AuthSHA1Pwd=#user,#password #command	
Parameter	#device	device specifier (see 5.1.2)
	#user	user name
	#password	for hidden password entry: string 'ask' (see 5.1.4 and below; hidden password entry is mandatory in FIPS-mode); otherwise: user password
	#command	command which has to be authenticated
Example	csadm AuthSHA1Pwd=paul,ask DeleteFile=xxx.mtc	
required state	*operational,* not available in any FIPS error state	
Output	none on success or error message	

For a CryptoServer in FIPS-mode, the usage of hidden password entry is mandatory.

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.

5.10 Secure Messaging

With Secure Messaging, commands to and from the CryptoServer will be encrypted and integrity protected using a Triple DES session key which was exchanged when opening the session.

See also 2.4.3 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.

5.10.1 SessionDH

This command opens a session for Secure Messaging using the Diffie-Hellman key agreement (according to [PKCS#3]). The session will be closed automatically when the command execution finishes.

Syntax	csadm [Dev=#device] <*Authentication*> SessionDH=#size #commands	
Parameters	#device	device specifier (see 5.1.2)
	#size	size of the Diffie-Hellman parameters (prime) in bits, recommended value is 1024
	#password	user password
	#commands	commands which should be executed within the session
Required State	*operational,* not available in any FIPS error state	
Authentication	authentication required	
Output	none on success or error message	
Note	The permissions of the user who has authenticated the session are granted to the whole session (i. e. to all commands following the *SessionDH* command in the same command line).	

5.11 Commands to Set Up the CryptoServer in Personalization Mode

This command group addresses a CryptoServer in personalization mode and *initialized* state. In this mode, the CryptoServer has not entered FIPS-mode yet, and the boot loader is the only active firmware (i. e. the module is in boot loader mode, see 2.3.4). The command group offers functionality for base administration during the process of setup and personalization of the CryptoServer before FIPS-mode is entered.

The commands described in this chapter are intended to be used during the CryptoServer's setup and personalization process which takes place before the module enters FIPS-mode.

If once the CryptoServer is in FIPS-mode, most of these commands are not available (exceptions are the commands for status requests, see below).

If the CryptoServer is in personalization mode (as it is e. g. after an alarm), the offered security relevant commands have to be signed with the private part of the *Initialization Key*.

The System Administrator (unnamed in personalization mode, but later called 'ADMIN' in FIPS-mode), who is owner of the Initialization Key*, is responsible for this part of administration.*

The following table shows all commands that are available in personalization mode and the required mode and authentication key of the CryptoServer:

Command	***Also available in FIPS mode (non-error state)?***	***Also available in FIPS mode and FIPS error state?***	***Required authentication key?***
GetState	yes	yes	none
StartOS	no	no	none
RecoverOS	no	no	none
BLChangeInitKey	no	no	Initialization Key
BLClear	no	no	Initialization Key
BLSetRTC	no	no	Initialization Key
BLResetAlarm	no	no	Initialization Key
GetAlarmLog	yes	yes	none
GetTempLog	yes	yes	none
GetTimeLog	yes	yes	none
GetBootLog	yes	yes	none

The commands

- GetState
- GetAlarmLog
- GetTempLog
- GetBootLog
- GetTimeLog

which request status information can be executed both in FIPS-mode and in personalization mode.
*If the CryptoServer is in FIPS-mode, these commands will furthermore be executed in normal operational state as well as in any FIPS Error state (*Boot Loader Error *state in boot loader mode as well as* OS Error *state in operational mode), see also chapter 5.7.*

5.11.1 GetState

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

The GetState *command is responded by the CryptoServer in any mode and state (in FIPS-mode as well as in personalization mode, and in normal operational state as well as in any* FIPS error *state) 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 5.1.2)
required state	any
Authentication	none
Output	Example for CryptoServer in in personalization mode and initialized state: **mode = Bootloader Mode** **state = INITIALIZED (0x00000004)** **temp = 35,5 [C]** **alarm = OFF** **bl_ver = 01020300** **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 5.7.1.

For further output examples see chapter 4.2.

5.11.2 BLClear

This command erases the CryptoServer to the *initialized* state (see also 3.5). This means that inside the CryptoServer the following data are erased:

- 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 fields adm1 and adm2 of the EID
- the Master Key $\mathbf{K_{CS2}}$.

This clear command is only available in personalization mode.
It should be executed …

- *before or after changing the* Initialization Key *(e. g. when switching from test to live operation), see 4.8,*
- *if the back-up copies of the base firmware modules shall be updated.*

Syntax	csadm [Dev=#device] InitPrvKey=#keyspec BLClear
Parameter	#device device specifier (see 5.1.2) #keyspec private part of the *Initialization Key* *(see 5.1.3 for possible key specifiers)*
Example	csadm InitPrvKey=c:\keys\init_prv.key BLClear
required state	*initialized* (personalization mode required)
Authentication	*Initialization Key*
Output	none on success or error message

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

This command is only available in personalization mode.

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 Utimaco's* Production Key*).*
Therefore in the process of changing the Initialization Key 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 deputy),*
- *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] InitPrvKey=#keyspec [Admin3=#value] ... … BLChangeInitKey=#keyspec
Parameter	#device device specifier (see 5.1.2) #keyspec InitPrvKey: private part of the old *Initialization Key* BLChangeInitKey: public part of the new *Initialization Key* *(see 5.1.3 for possible key specifiers)* #value 16 bytes string (should be name of new *Initialization Key*)
Example	csadm InitPrvKey=c:\keys\init_old_prv.key Admin3='Init-Live01-Key' ... … BLChangeInitKey=c:\keys\init_new_pub.key
required state	*initialized* (personalization 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 changing the *Initialization Key* all data inside the CryptoServer have to be cleared (*BLClear* command, chapter 5.11.2). ■ 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 4.7) and again loaded onto the CryptoServer.

5.11.4 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.

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 available in FIPS-mode (see 5.7.3). 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 5.11.2) and reload all wanted files using *BLLoadFile*.

This command is only available in personalization mode.
Unlike the LoadFile *command (see 5.7.3) that is offered in FIPS-mode the* BLLoadFile *command*

- *should only be used to load the most important firmware modules (SMOS, CMDS, UTIL and ADM), needed to get a minimal working system*
- *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).*

During the process of setup and personalization of the CryptoServer in order to get it into FIPS-mode later, the BLLoadFile *command must and should only be used to load the firmware modules SMOS, CMDS, UTIL and ADM in the respective versions listed in chapter 7 'Appendix: Mandatory Firmware Modules'.*

Syntax	csadm	[Dev=#device] InitPrvKey=#keyspec BLLoadFile=#file
Parameter	#device	device specifier (see 5.1.2)
	#keyspec	private part of the *Initialization Key* (see 5.1.3)
	#file	any file, usually a firmware module
Example	csadm InitPrvKey=:cs2:cp8:COM1 … … BLLoadFile=c:\firmware\release\exmp.mtc	
required state	*initialized* (personalization 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.

5.11.5 BLSetRTC

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

This command is only available in personalization mode.
In FIPS-mode, the SetTime *command has to be used to set the CryptoServer's clock, see 5.7.6. From the user's point of view there is no difference between these two commands, except that* BLSetRTC *can only be performed in personalization mode whereas* SetTime *can only be performed FIPS-mode (and operational state).*

Syntax	csadm [Dev=#device] InitPrvKey=#keyspec BLSetRTC=#time
Parameter	#device device specifier (see 5.1.2) #keyspec InitPrvKey: private part of the *Initialization Key (see 5.1.3 for possible key specifiers)* #time 'YYYYMMDDHHMMSS' or 'SYSTEM'
Example	■ csadm InitPrvKey=c:\keys\init_prv.key BLSetRTC=20011206115530 ■ csadm InitPrvKey=:cs2:cp8:COM1 BLSetRTC=SYSTEM
required state	*initialized* (personalization mode required)
Authentication	*Initialization Key*
Output	none on success or error message
Note	■ The time format is BCD coded: YYYYMMDDHHMMSS ■ If SYSTEM is given as argument the host PC system time is used. ■ 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 chapters 5.7.11 and 5.11.9) 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.

5.11.6 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:

This command is only available in personalization mode (but, after any alarm has occured, the CryptoServer will automatically be in personalization mode). If the BLResetAlarm *command has been performed and the alarm state is reset successfully, the CryptoServer will remain in personalization mode and thus only the commands described in this chapter can be performed.*

After an alarm, except for GetState *no other CryptoServer command can be executed until the* BLResetAlarm *command has been executed successfully. The* GetState *command (see chapter 5.11.1) shows detailed information about the actual 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 6.2 *Alarm Treatment* for more details about what to do in case of an alarm.

Syntax	csadm [Dev=#device] InitPrvKey=#keyspec BLResetAlarm
Parameter	#device device specifier (see 5.1.2) #keyspec InitPrvKey: private part of the *Initialization Key* *(see 5.1.3 for possible key specifiers)*
Example	csadm InitPrvKey=c:\keys\init_prv.key BLResetAlarm
required state	*initialized* (personalization mode required)
Authentication	*Initialization 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 5.7.9 and 5.11.7) can be used to view this file.

5.11.7 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. If no alarm has occurred since the CryptoServer's production, the file 'alarm.log' does not exist.

This command is responded by the CryptoServer in both personalization mode and FIPS-mode.

Syntax	csadm [Dev=#device] GetAlarmLog
Parameter	#device device specifier (see 5.1.2)
required state	*initialized* or *operational* (command can be performed in both personalization mode and FIPS-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 the file does not exist 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 is executed (manually) ■ inval_MK Invalid (corrupted) CryptoServer Master Key $\mathbf{K_{CS2}}$ (this usually occurs in case of an empty battery) See chapter 6.2 '*Alarm Treatment*' for how to deal with an alarm

5.11.8 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*). The 'temp.log' file is deleted when the CryptoServer is cleared to the *initialized* state (see *BLClear*, chapter 5.11.2). 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 gets 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 CryptoServer's temperature treatment.

This command is responded by the CryptoServer in both personalization mode and FIPS-mode.

Syntax	csadm [Dev=#device] GetTempLog
Parameter	#device device specifier (see 5.1.2)
required state	*initialized* or *operational* (command can be performed in both personalization mode and FIPS-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 'temp.log' file does not exist an error message is returned.

5.11.9 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. The 'time.log' file is not write-protected. It will be deleted whenever the CryptoServer is completely cleared (see *BLClear*, chapter 5.11.2) and can also explicitly be deleted by the user (see *DeleteFile* command chapter 5.7.4). 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 if the CryptoServer is in personalization mode (see BLSetRTC*, chapter 5.11.5) as well as by the administration module ADM if the CryptoServer is in FIPS-mode (see* SetTime, *chapter 5.7.6).*

This command is responded by the CryptoServer in both personalization mode and FIPS-mode.

Syntax	csadm [Dev=#device] GetTimeLog
Parameter	#device device specifier (see 5.1.2)
required state	*initialized* or *operational* (command can be performed in both personalization mode and FIPS-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 file does not exist, an error message will be returned.

5.11.10 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.

This command is only available in personalization mode.
In personalization mode, 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 does not receive any command within 10 seconds (see chapter 2.3.3.1)*

Syntax	csadm [Dev=#device] StartOS
Parameter	#device device specifier (see 5.1.2)
Authentication	none
required state	*initialized* (personalization mode required)
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 5.7.8).

This command can only be performed in personalization mode and initialized *state. After it is successfully performed, the CryptoServer is in the* operational *state and works in (normal) operational mode (see 2.3.4). If additionally the necessary firmware modules are loaded (see chapter 7: Appendix: Mandatory Firmware Modules), the CryptoServer will be in FIPS-mode then.*

5.11.11 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 5.11.4). It is not possible that all FIPS-relevant firmware is stored in the \SYS directory.

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

This command is intended to be used in case of problems with the StartOS *command (see above).*
The command can not be used to get the CryptoServer from personalization mode to FIPS-mode!

Syntax	csadm [Dev=#device] RecoverOS
Parameter	#device device specifier (see 5.1.2)
Authentication	none
required state	*initialized* (personalization mode required)
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 5.7.8).

This command can only be performed in personalization mode and initialized *state. After it is successfully performed, the CryptoServer is in the* operational *state and works in (failsafe) operational mode (see 2.3.4). But the CryptoServer is still in personalization mode and is NOT in FIPS-mode!*

5.12 Miscellaneous Commands

5.12.1 Sleep

The *Sleep* command delays the further command processing by the time given.

Syntax	csadm Sleep=#time
Parameter	#time delay time 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.

5.12.2 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] [<*Authentication*>] Cmd=#fc,#sfc,#$byte_1$,#$byte_2$,#$byte_3$,...	
Parameter	#device	device specifier (see 5.1.2)
	#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 5.12.3.*

5.12.3 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	# hex 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] [<*Authentication*>] CmdFile=#file
Parameter	#device device specifier (see 5.1.2) #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

5.12.4 SaveKey

This command reads a public part of the *Initialization Key* from a smart card and stores it as key file ('*.key').

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 InitPubKey=#keyspec SaveKey=#keyoutfile
Parameter	#keyspec key specifier (only key on smart card, see 5.1.3) #keyoutfile output file to store the key ('*.key').
Example	csadm InitPubKey=:cs2:cp8:COM1 SaveKey=C:\my_keys\pubkey1.key
Output	none on success or error message

5.12.5 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 5.1.3)
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.*

6 Troubleshooting

6.1 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.

What to do?

1. Perform the *GetState* command (see 5.7.1).

Result	Explanation / Reason / Adjustment
state = OPERATIONAL, FIPS mode = ON, CryptoServer is not in FIPS error state	Everything is ok. End.
state = OPERATIONAL, but CryptoServer is not in FIPS-mode	You may analyze the problem using the *ListModulesActive* and *GetBootLog* commands: Check if all necessary firmware modules are present and successfully initialized (see chapter 7 for a complete list). After that the CryptoServer should be set-up and personalized again, see 4.4.
state OPERATIONAL, FIPS mode = ON, CryptoServer is in FIPS error state	Quit error state, see 4.3.
state = INITIALIZED, ALARM = off, but CryptoServer is not in FIPS-mode	CryptoServer is in personalization mode or not correctly initialized in FIPS mode. ⇒ Re-initialize the CryptoServer, see 4.4.
state = INITIALIZED, ALARM = off, FIPS mode = ON, CryptoServer is in FIPS error state	Quit error state, see 4.3.

state INITIALIZED, ALARM = ON	an alarm has occurred (and is possibly physically still present) ⇒ see chapter 6.2 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 reset (2)
other errors: B901xxxx or B902xxxx	No connection to CryptoServer: communication problem, wrong host or device name, problem with network. ⇒ check parameters
No answer from CryptoServer	⇒ try a reset (2).

2. Perform the *Reset* command (see 5.6.1). Wait a minute, then perform a *GetState* command.

Result	**Explanation / Reason / Adjustment**
no error	Ok ⇒ back to (1)
any error	⇒ power-cycle the CryptoServer (3)

3. Remove the power from the CryptoServer for at least 30 seconds and power-up the CryptoServer again. Then perform a *GetState* command.

Result	**Explanation / Reason / Adjustment**
no error	Ok ⇒ back to (1)
any error	hardware problem ⇒ please contact manufacturer/Utimaco

6.2 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 5.7.1. 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: after any alarm the CryptoServer remains in personalization mode (i. e. it has left FIPS mode) and in global state *initialized*.

What to do?

1. If the alarm is physically still present: Remove the alarm cause if possible. Execute *GetState* again (see chapter 5.7.1) 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 'alarm has occurred').
2. If the alarm is still shown as 'present': please contact the manufacturer/Utimaco.
3. Else: Perform the *BLResetAlarm* command (see chapter 5.11.6).
4. Execute *GetState* again. The alarm state should now be 'OFF'.
5. Since the alarm has cleared all data and firmware modules, and in order to enter FIPS mode again, the CryptoServer must be re-personalized (see chapter 4.4).

7 Appendix: Mandatory Firmware Modules

The following firmware modules (in MTC format) are mandatory in FIPS-mode and have to be loaded by the CryptoServer's *System Administrator* during the setup and personalization process (see 4.4):

- SMOS: file 'smos.mtc', version 2.0.0.6
- CMDS: file 'cmds.mtc', version 1.0.9.0
- UTIL: file 'util.mtc', version 2.0.0.3
- ADM: file 'adm.mtc', version 2.0.0.2
- DB: file 'db.mtc', version 1.0.1.2
- VDES: file 'vdes.mtc', version 1.0.1.1
- VRSA: file 'vrsa.mtc', version 1.0.5.4
- AES: file 'aes.mtc', version 1.0.3.0
- HASH: file 'hash.mtc', version 1.0.4.1
- LNA: file 'lna.mtc', version 1.0.4.0
- ASN1: file 'asn1.mtc', version 1.0.3.2
- ECA: file 'eca.mtc', version 1.0.0.0
- ECDSA: file 'ecdsa.mtc', version 1.0.0.0
- CSI: file 'csi.mtc', version 1.2.1.1
- FIPS140: file 'fips140.mtc', version 2.0.0.1

The listed firmware modules are approved in the context of the FIPS 140-2 validation process of the CryptoServer. These firmware modules have to be loaded if the CryptoServer shall run in FIPS mode.
If the CryptoServer is run with other firmware than the above listed MTC modules, or if the listed firmware is used in other versions, the CryptoServer can not considered to be in certified FIPS mode!

The FIPS140-2 approved version of the boot loader firmware is version 2.0.2.4. The boot loader firmware module is loaded by Utimaco during the CryptoServer's production process and cannot be changed or deleted by the customer.

8 References

No.	Title/Company	Doc.-No.
[ANSIX9.31]	ANSI X9.31-1998: Digital Signatures using Reversible Public Key Cryptography for the Financial Services Industry (rDSA) / American Bankers Association, 1998	
[CSCSI]	CryptoServer - Cryptographic Services Interface – Firmware Module CSI – Interface Specification / Utimaco Safeware AG	2004-0003
[CSFIPS-UserGuide]	CryptoServer CS – User's Guide for CryptoServer in FIPS-Mode / Utimaco SafewareAG	2004-0005
[CSInstall-Manual]	CryptoServer – Installation Manual / Utimaco SafewareAG	2003-0007
[FIPS186-2]	FIPS PUB 186-2: Digital Signature Standard (DSS) / 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	