

# CryptoServer

csadm Manual

## Imprint

Copyright 2024	Utimaco IS GmbH Germanusstr. 4 D-52080 Aachen Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226) EMEA +49 800-627-3081 APAC +81 800-919-1301
Internet e-mail	<a href="https://support.hsm.utimaco.com/">https://support.hsm.utimaco.com/</a> <a href="mailto:support@utimaco.com">support@utimaco.com</a>
Document Version	2.11.16
Product Version	6.0.0
Date	2024-10-23
Document No.	2009-0003
Status	<b>PUBLISHED</b>

All rights reserved	<p>No part of this documentation may be reproduced in any form (printing, photocopy or according to any other process) without the written approval of Utimaco IS GmbH or be processed, reproduced or distributed using electronic systems.</p> <p>Utimaco IS GmbH reserves the right to modify or amend the documentation at any time without prior notice. Utimaco IS GmbH assumes no liability for typographical errors and damages incurred due to them. Any mention of the company name Utimaco in this documents refers to the Utimaco IS GmbH.</p> <p>All trademarks and registered trademarks are the property of their respective owners.</p>
---------------------	--

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>8</b>
1.1	Other Manuals .....	8
1.2	Document Conventions .....	10
<b>2</b>	<b>Installation.....</b>	<b>12</b>
2.1	Optional Configuration .....	14
2.1.1	Setting, Verifying and Deleting CRYPTOSERVER on Windows .....	14
2.1.2	Setting, Verifying and Deleting CRYPTOSERVER on Linux .....	16
<b>3</b>	<b>Security Management .....</b>	<b>18</b>
3.1	Maximum for Failed Authentication Attempts .....	18
3.2	Secure Messaging .....	18
3.3	Auditing .....	21
3.3.1	Configuring Auditable Events .....	21
3.3.2	Signed Audit Logs .....	23
3.4	Signed Configuration Files .....	24
3.4.1	Sample Configuration File cmds_sample.cfg .....	26
3.4.2	Interface Hardening by Disabling Selected Functions.....	26
3.4.3	Configurable Role-based Access.....	27
3.4.4	List of External Firmware Functions.....	29
<b>4</b>	<b>Administration of CryptoServer with csadm .....</b>	<b>30</b>
4.1	Overview of csadm .....	30
4.1.1	Syntax of csadm.....	30
4.1.2	Command Execution with the csadm Tool .....	33
4.1.3	Password Authentication .....	35
4.1.4	Storage and Specification of RSA and ECDSA Keys for Authentication.....	36
4.1.4.1	Key Specifiers.....	37
4.1.4.2	Smartcard Specifiers.....	38
4.2	Basic Commands.....	39
4.2.1	Help .....	40
4.2.2	More .....	40
4.2.3	PrintError.....	41
4.2.4	StrError.....	41
4.2.5	Version .....	42
4.2.6	SetTimeout.....	42

---

4.2.7	Cmd .....	43
4.2.8	CmdFile .....	44
4.2.9	Sleep.....	45
4.2.10	ConnTimeout .....	46
4.3	Commands for Authentication .....	46
4.3.1	LogonSign .....	48
4.3.2	LogonPass.....	51
4.3.3	ShowAuthState.....	53
4.3.4	GetHSMAuthKey.....	54
4.4	Commands for Administration .....	55
4.4.1	GetState .....	56
4.4.2	SetAdminMode .....	61
4.4.3	SetStartupMode .....	62
4.4.4	GetStartupMode .....	63
4.4.5	GetBattState.....	64
4.4.6	StartOS .....	65
4.4.7	RecoverOS .....	66
4.4.8	GetTime .....	67
4.4.9	SetTime .....	67
4.4.10	GetBootLog .....	69
4.4.11	GetAuditLog.....	71
4.4.12	ClearAuditLog .....	71
4.4.13	GetAuditConfig .....	72
4.4.14	SetAuditConfig .....	73
4.4.15	GenerateAuditLogKey.....	75
4.4.16	GetAuditLogKey.....	77
4.4.17	GetSignedAuditLog.....	78
4.4.18	VerifySignedAuditLog.....	80
4.4.19	ClearAuditLogFiles .....	81
4.4.20	MemInfo .....	83
4.4.21	Test.....	84
4.4.22	ResetAlarm .....	86
4.4.23	GetModel .....	87
4.4.24	Reset .....	87

4.4.25	ResetToBL .....	88
4.4.26	Restart .....	91
4.4.27	GetInfo .....	92
4.4.28	MemInfoCSV .....	94
4.4.29	RamInfo .....	94
4.4.30	RamInfoCSV .....	95
4.4.31	GenRandom .....	95
4.5	Commands for Managing the User Authentication Keys .....	96
4.5.1	GenKey .....	96
4.5.2	SaveKey .....	99
4.5.3	BackupKey .....	101
4.5.4	CopyBackupCard .....	102
4.5.5	GetCardInfo .....	103
4.5.6	ChangePassword .....	104
4.5.7	ChangePIN .....	106
4.5.8	BackupDatabase .....	107
4.5.9	RestoreDatabase .....	108
4.6	Commands for User Management .....	111
4.6.1	ListUser .....	114
4.6.2	AddUser .....	117
4.6.2.1	Adding PKCS#11 Security Officers with RSA Signature Authentication with a Smartcard .....	122
4.6.3	ChangeUser .....	124
4.6.4	DeleteUser .....	127
4.6.5	BackupUser .....	128
4.6.6	RestoreUser .....	130
4.6.7	SetMaxAuthFails .....	135
4.6.8	GetMaxAuthFails .....	136
4.7	Commands for Managing the Master Backup Keys .....	137
4.7.1	MBKListKeys .....	139
4.7.2	MBKGenerateKey .....	140
4.7.3	MBKImportKey .....	142
4.7.4	MBKCopyKey .....	145
4.7.5	MBKCardInfo .....	147
4.7.6	MBKCardCopy .....	148
4.7.7	MBKPINChange .....	149

---

4.8	Commands for Firmware Management .....	149
4.8.1	ListFirmware.....	149
4.8.2	ListFiles .....	153
4.8.3	LoadFile.....	157
4.8.4	LoadPkg.....	160
4.8.5	DeleteFile.....	163
4.8.6	Pack.....	165
4.8.7	Unpack.....	165
4.8.8	ListPkg .....	166
4.8.9	CheckPkg.....	169
4.8.10	ModuleInfo .....	170
4.8.11	Clear .....	172
4.9	Commands for Firmware Packaging .....	174
4.9.1	MakeMTC.....	176
4.9.2	RemoveMTC .....	178
4.9.3	VerifyMTC.....	179
4.9.4	VerifyPkg.....	180
4.9.5	RenameToVersion .....	182
4.9.6	LoadFWDeckKey .....	183
4.9.7	LoadAltMdlSigKey .....	184
4.9.8	SignConfig.....	185
4.10	Commands for Administration of CryptoServer LAN.....	187
4.10.1	CSLGetConnections.....	187
4.10.2	CSLScanDevices.....	188
4.10.3	CSLGetVersion.....	190
4.10.4	CSLGetStatus .....	190
4.10.5	CSLGetLogFile .....	191
4.10.6	CSLGetConfigFile .....	192
4.10.7	CSLPutConfigFile .....	193
4.10.8	CSLSetTracelevel .....	194
4.10.9	CSLShutdown.....	195
4.10.10	CSLReboot.....	196
4.10.11	CSLGetTime.....	196
4.10.12	CSLSetTime .....	197

4.10.13	CSLGetSerial.....	198
4.10.14	CSLGetLoad.....	199
4.10.15	CSLListPPApps.....	200
<b>5</b>	<b>Basic Administration Tasks .....</b>	<b>202</b>
5.1	Setting Up a New CryptoServer .....	202
5.2	Entering and Leaving the 'Administration only' Mode.....	206
5.3	Generating a User Authentication Key.....	207
5.4	Checking the Battery State.....	209
5.5	Performing a Complete Clear of the CryptoServer.....	210
5.6	Gathering Diagnostic Information .....	214
<b>6</b>	<b>Advanced Administration Tasks .....</b>	<b>216</b>
6.1	Creating Self-Signed/Encrypted Firmware Modules.....	216
6.2	Loading a Self-Signed Encrypted Firmware Module into the CryptoServer .....	218
6.3	Creating and Using the Signed Configuration File cmds.scf .....	222
6.4	Generating and Verifying Signed Audit Log Files .....	225
6.5	Enabling Mutual Authentication .....	228
<b>7</b>	<b>Troubleshooting .....</b>	<b>233</b>
7.1	Checking the Operativeness and the State of the CryptoServer .....	233
7.2	Alarm Treatment .....	235
7.3	Leaving Error State.....	237
<b>8</b>	<b>Contact Address for Support Queries .....</b>	<b>242</b>
<b>9</b>	<b>References .....</b>	<b>243</b>

# 1 Introduction

Thank you for purchasing our CryptoServer security system. We hope you are satisfied with our product. Please do not hesitate to contact us if you have any questions or comments.

The CryptoServer command-line tool csadm is a program designed for the administration of the device that can be called from either a command line or a batch file. It handles all typical administration tasks like setup, status monitoring, managing users, firmware, and keys. Additionally, it can perform advanced administration functions that are exclusively available for customers working with SDK devices that want to extend the standard functionality of the device with self-developed firmware modules providing specific cryptographic functions and commands.

All operating systems that are currently supported for the csadm host computer are listed in the release notes.

## 1.1 Other Manuals

The CryptoServer is supplied as a PCI-Express (PCIe) plug-in card in the following series:

- CryptoServer CSe-Series
- CryptoServer Se-Series Gen2

The CryptoServer LAN (appliance) is supplied in the following series:

- CryptoServer LAN CSe-Series
- CryptoServer LAN Se-Series Gen2

We provide the following manuals on the product CD for all series CryptoServer cards and for all series CryptoServer LAN (appliance):

### Quick Start Guides

You will find these Manuals in the main directory of the SecurityServer product CD. They are available only in English, do not cover all possible scenarios, and are intended as a supplement to the product documentation provided on the SecurityServer product CD.



- *CryptoServer LAN V5 - Quick Start Guide*

This guide provides step-by-step instructions on how to bring the CryptoServer LAN into service, how to prepare a computer (Windows 7) for the CryptoServer administration and how to start administrating your CryptoServer with the Java-based GUI CryptoServer Administration Tool (CAT).

- *CryptoServer PCIe - Quick Start Guide for Linux*

This guide provides a step-by-step instruction on how to bring the CryptoServer PCIe card into service, how to install the CryptoServer driver on a computer with minimal RHEL 7.0 installation and how to start administrating your CryptoServer with the CryptoServer Command-line Administration Tool (csadm).

- *CryptoServer PCIe - Quick Start Guide for Windows*

This guide provides for step-by-step instructions on how to bring the CryptoServer PCIe card into service, how to install the CryptoServer driver on a Windows computer and how to start administrating your CryptoServer with the CryptoServer Command-line Administration Tool (csadm).

## Manuals for System Administrators

You will find the administration manuals on the product CD in the following directory: ...  
Documentation\Administration Guides\

- *CryptoServer – Administration Manual*

This manual provides provides a detailed description of the CryptoServer functionality, concepts and security mechanisms. It also describes CryptoServer setup and maintenance.

- *CryptoServer LAN V5 – Administration Manual*

This manual describes how to administer a CryptoServer LAN appliance by using the front panel of the appliance.

- *CryptoServer - Troubleshooting*

If problems occur during the use of the PCIe card or a LAN (appliance), read this manual.

- *CryptoServer - PKCS#11 P11CAT - Manual*

If you need to administer the PKCS#11 R3 interface with the PKCS#11 CryptoServer Administration Tool (P11CAT), read this manual.

## Operating Manual

You will find these manuals on the product CD in the following directory: ...  
Documentation\Operating Manuals\ . In these manuals, you find all the necessary information for using appropriately the CryptoServer PCIe card hardware and the CryptoServer LAN (appliance) hardware.

## 1.2 Document Conventions

We use the following document conventions:

<b>Convention</b>	<b>Use</b>	<b>Example</b>
<b>Bold</b>	Items of the Graphical User Interface (GUI), e.g., menu options	Press <b>OK</b>
<b>Monospaced</b>	Code that is given for explanation or as an example, file paths	<code>chsm-create</code>
<i>Italic</i>	References and important terms	See <i>Sample Chapter</i> in the <i>CryptoServer - Sample Manual</i>

Table 1: Document conventions

We use special icons to highlight the most important notes and information.



Here, you find important safety information that should be followed.



Here, you find additional notes or supplementary information.



This message marks the result expected after the successful execution of an instruction.

## 2 Installation

The following requirements have to be fulfilled prior to the installation of csadm.

### Hardware requirements (PC):

- No special requirements as far as memory and CPU performance are concerned.
- Network interface card to access LAN device.
- A free USB port is needed to connect the PIN pad (smartcard reader with keyboard and display).

### Software requirements (OS):

You will find the list of all currently supported operating systems in the document `CS_PD_SecurityServer_Supported_Platforms.pdf` on the SecurityServer product CD in the directory `...\Documentation\Product Details`.

This section describes how to install csadm on an administration computer. The csadm installation file is provided on the SecurityServer product bundle.

### Installing csadm on a computer with a Windows operating system

On an administration computer running a Windows operating system csadm is installed by default during the installation of the CryptoServer software provided on the SecurityServer bundle.

The following steps describe how to install csadm on a Windows host computer without using the CryptoServer installer file, `SecurityServer-<version>.msi`, provided on the SecurityServer bundle.

You find the csadm installation file for Windows, `csadm.exe`, here:

- For Windows 64-bit operating systems  
`Software\Windows\Administration\`

This directory is to be found on the general SecurityServer bundle.

1. Copy the EXE file to a well-chosen directory.
2. Add this directory to the `PATH` environment variable to be able to call the Administration Tool from any other directory.

3. If you do not want to set the `Dev=` parameter with each execution of a `csadm` command:  
It is possible to set an environment variable `CRYPTOSERVER` (with the value `PCI:n` where  $n = \{0, 1, 2 \dots, 31\}$ ) which sets the CryptoServer address permanently (unless a `Dev=` parameter is explicitly set for a specific command).
4. Verify whether mutual authentication should be enabled. If you do not want to perform mutual authentication, perform the following substeps to ensure that the `CS_AUTH_KEYS` system environment variable is not available.  
Either  
open **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment Variables > System variables**, select `CS_AUTH_KEYS`, if available, and click **Delete**,  
or  
open **Start > All Programs > Accessories > Command Prompt > Run as administrator**, and perform the following command:

```
reg delete "HKLM\SYSTEM\CurrentControlSet\Control\Session  
Manager\Environment" /F /V CS_AUTH_KEYS
```

### Installing csadm on a computer with a UNIX-like operating system

You find the `csadm` installation file for UNIX-like operating systems, `csadm`, here:

- For Linux 32-bit operating systems  
`Software/Linux/x86-32/Administration/`  
This directory is to be found on the 32-bit add-on bundle for the SecurityServer.
- For Linux 64-bit operating systems  
`Software/Linux/Administration/`  
This directory is to be found on the general SecurityServer bundle.

1. Copy the executable file `csadm` to a well-chosen `bin` directory.
2. If you do not want to set the `Dev=` parameter with each execution of a `csadm` command. It is possible to set an environment variable `CRYPTOSERVER` (e.g., with the value `/dev/cs2.n` where  $n = \{0, 1, 2 \dots, 7\}$ ), which defines the CryptoServer address permanently (unless a `Dev=` parameter is explicitly set for a specific command).
3. Verify whether mutual authentication should be enabled. If you do not want to perform mutual authentication, perform the following sub-steps to ensure that the `CS_AUTH_KEYS` system environment variable is not available.  
Example:

- a. Make sure that you are in the home directory of a user, for example, `/home/user01`.
- b. Remove the line starting with `export CS_AUTH_KEYS` from the `/home/user01/.bashrc`  
Example: `export CS_AUTH_KEYS=~/.HSMauth.key`
- c. Perform the following command to remove `CS_AUTH_KEYS` in the current command line:  
`unset CS_AUTH_KEYS`

In other Linux shells, the commands might be different.

## 2.1 Optional Configuration

For setting up and administering the device, the command-line tool `csadm` requires the address of the device to connect to. You can set this address by specifying it for every command in the `Dev` command parameter.

Example:

```
csadm Dev=192.168.1.1 GetState
```

Optionally and for more convenience, you can set the address of your device in the `CRYPTOSERVER` environment variable. In this case, the `Dev` parameter in the `csadm` command can be omitted.

The `CRYPTOSERVER` environment variable is not a default for the following parameters.

- `Device` parameter in the `cfg` file  
This parameter is used for actions initiated by P11CAT or p11tool2.
- `Device` parameter in the `cfg` file  
This parameter is used for actions initiated by CSP/CNG.
- `Device` parameter in the `conf` file  
This parameter is used for some communication of the CryptoServer LAN.

### 2.1.1 Setting, Verifying and Deleting CRYPTOSERVER on Windows

Setting CRYPTOSERVER

- For the current command-line

Example:

```
set CRYPTOSERVER=192.168.1.1
```

- For all future command-lines

- Using an administrator command-line

- **Start > All Programs > Accessories > Right mouse click on Command Prompt > Run as administrator**

- Perform a setx command

Example: `setx CRYPTOSERVER 192.168.1.1 /m`

The /m parameter indicates that the environment variable is a system environment variable and no user environment variable. The value is only set for future command-lines but not for the current one.

- Using the GUI

- Ensure that you are logged in as an administrator.
- **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment variables > System variables > New or select CRYPTOSERVER > Edit plus OK**

## Verifying CRYPTOSERVER

- For the current command-line

Example: `set CRYPTOSERVER`

- For all future command-lines

- Ensure that you are logged in as an administrator.
- **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment variables > System variables > CRYPTOSERVER**

## Deleting CRYPTOSERVER

- For the current command-line

Example:

```
set CRYPTOSERVER=
```

- For all future command-lines
- Using an administrator command-line
  - **Start > All Programs > Accessories >** Right mouse click on Command Prompt > **Run as administrator**

- Perform a setx

Example:

```
setx CRYPTOSERVER "" /m
```

The /m parameter indicates that the environment variable is a system environment variable and no user environment variable. The value is only set for future command-lines but not for the current one.

- Using the GUI
  - Ensure that you are logged in as an administrator.
  - **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment variables > System variables > CRYPTOSERVER > Delete**

## 2.1.2 Setting, Verifying and Deleting CRYPTOSERVER on Linux



The commands in this section are only examples. Depending on your shell, other commands might be suitable.

### Setting CRYPTOSERVER



- For the current command line

Example:

```
export CRYPTOSERVER=192.168.1.1
```

- For all future command-lines

- a. Append an export command to the `.bashrc`



It is very important that you use `>>` in the following command. It appends the preceding text to the specified file. Do not use `>`, because it overwrites the entire file.

Example:

```
echo export CRYPTOSERVER=192.168.1.1 >> ~/.bashrc
```

- b. Apply the changes to the current command-line as well by performing the following command.

Example: `. ~/.bashrc`

## Verifying CRYPTOSERVER

Perform one of the following commands to verify the value of the CRYPTOSERVER environment variable.

- For the current command line

Example 1: `env | grep CRYPTOSERVER`

Example 2: `printenv | grep CRYPTOSERVER`

Example 3: `echo $CRYPTOSERVER`

## Deleting CRYPTOSERVER

- For the current command-line

Example:

```
export CRYPTOSERVER=
```

- For all future command-lines

Remove the following line from the `~/.bashrc` file.

Example:

```
export CRYPTOSERVER=192.168.1.1
```

## 3 Security Management

This chapter describes the various security mechanisms of the CryptoServer, like, for example, user authentication, secure messaging, or alarm treatment.

### 3.1 Maximum for Failed Authentication Attempts

In the device, the number of consecutive failed authentication attempts for every user is automatically counted and stored as a user attribute (counter for failed authentication attempts, *AuthenticationFailureCounter*, denoted as  $Z[n]$ ). If the authentication of a user fails, the *AuthenticationFailureCounter* for the user is incremented by one. On successful authentication the *AuthenticationFailureCounter* is reset to zero.

A maximum value for the number of failed authentication attempts (*MaxAuthFails*) can be set by using the csadm command `SetMaxAuthFails`, where  $0 \leq \text{MaxAuthFails} \leq 255$ . To retrieve the defined value for *MaxAuthFails* the csadm command `GetMaxAuthFails` should be used. By default, *MaxAuthFails* = 0 which means that the number of failed authentication attempts for all device users is not limited. Any other value for *MaxAuthFails* means that for each user there are only *MaxAuthFails*-1 consecutive failed authentication attempts allowed. The user is locked automatically after *MaxAuthFails* consecutive failed authentication attempts, i.e., if the *AuthenticationFailureCounter* of the user equals the value of *MaxAuthFails*.

If a user is locked no further authentication is possible for her/him, consequently this user cannot perform any command which has to be authenticated. Only a user with user management permission (permission 2 in user group 7, 20000000) can unlock a locked user by setting her/his *AuthenticationFailureCounter* back to zero by using the csadm command `ChangeUser`.

### 3.2 Secure Messaging

The device supports Secure Messaging for communications between the host and itself. Commands from the host to the device and the replies to the host can be AES encrypted and the integrity of the data can be protected by a AES MAC (Message Authentication Code). For this purpose, a secure messaging header data block is added to the command and the answer data block. The detailed structure of the header data blocks is described in *CryptoServer - Firmware Module CMDS - Interface Specification* provided within the product bundle.

In Secure Messaging between the device and the host, a new random session key is generated for each connection. This prevents recorded data packets belonging to an earlier

connection from being imported into a new connection. No valid data packets can be present once the imported data packet has been decrypted with the current session key.

In addition, the device generates a start value for a sequence counter and a session ID for every new session key. The sequence counter increases every time a command is sent and is also included in the MAC calculation or integrity check. This ensures that no commands are recorded within the same connection and sent to the device again. The unique session ID guarantees that every session key is uniquely identifiable. All commands that use the same session key and session ID are part of the same session (connection).



The SecurityServer/CryptoServer SDK 4.10 and later support a maximum of 4096 session keys (connections) active at the same time. If another session key is requested for a connection, exceeding the maximum of 4096, the oldest connection is closed and the session key, associated with it, becomes invalid

The SecurityServer/CryptoServer SDK 4.01 and earlier support a maximum of 256 session keys which are active at the same time and used by different host applications (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 the session key, associated with it, becomes invalid.



On FIPS devices, Secure Messaging must be used for every command which has to be authenticated.

To use the secure messaging functionality, the csadm commands `csadm LogonSign` and `csadm LogonPass`, are used together in one command line with the command(s) for which Secure Messaging should be used. csadm will automatically open an authenticated Secure Messaging session, perform the specific command(s) with Secure Messaging (i.e., encrypted and MAC-secured) and close the Secure Messaging session on the device again. The host and the device agree upon an AES session key. The external device function `GetSessionKey` is called to generate an AES session key.

1. The session key can be negotiated in one of the following ways:
  - With the Ephemeral Unified Model Key Agreement Scheme according to SP800-56A r3 (FIPS-compliant; based on EC Diffie Hellman). This is automatically tried first. If

the device does not support this, the Diffie-Hellman key establishment protocol is used.

- With the Diffie-Hellman key establishment protocol



On FIPS devices the Diffie-Hellman key establishment protocol is blocked.

2. The host and the device exchange encrypted commands in a Secure Messaging session. The host can now send commands to the device that are AES-encrypted and integrity-protected with an AES MAC. The respective answers to these commands that are sent back to the host by the device are always encrypted and protected with a MAC, too. The sequence counter is used as initialization vector for the AES encryption and MAC calculation and is incremented after every command to prevent unauthorized replays of the commands.



The session key is identified by a session ID. All commands using the same session ID and the same session key belong to the same session. This way a secure channel can be established between the device and the host application using the Secure Messaging mechanism.

The device accepts commands that do not require authentication (i.e., in cleartext), sent to it over an unprotected connection, in parallel to commands sent to it within a protected Secure Messaging session. If the device receives an encrypted command (i.e., a command using the Secure Messaging layer of the protocol stack of the device), it checks the MAC. The command is rejected if the MAC is invalid.



A session key automatically becomes invalid if it is not used for 15 minutes. The device supports a maximum of 4096 session keys that can be active simultaneously and can be used by different host applications (each key identified by its session ID). If a host application requests a new session key while the maximum of 4096 sessions are already active, the oldest session is closed and the session key associated with it becomes invalid.

If the `GetSessionKey` function is authenticated by one or more users using single command authentication, the permissions of the corresponding user(s) will be granted to the established Secure Messaging session. All commands that are protected with this session

key have the permissions of these users without any extra authentication being necessary. However, outside of this session, the former authentication status is preserved.

## 3.3 Auditing

In the audit log all events and actions involving security issues that occur or are performed whilst the CryptoServer is running are recorded.



For a detailed overview of audit log entries, see *Audit Log Entries* in the [CryptoServer - Administration Manual](#) (p. 243).

### 3.3.1 Configuring Auditable Events

The CryptoServer offers extensive audit functionality.

CryptoServer audit log files can be read at any time and in any mode with the `csadm GetAuditLog` command, see [GetAuditLog](#) (p. 71). A CryptoServer audit log file can be deleted with the `csadm ClearAuditLog` command, see [ClearAuditLog](#) (p. 71). The current configuration of the CryptoServer audit functionality can be retrieved with the `csadm GetAuditConfig` command, see [GetAuditConfig](#) (p. 72), and it can be changed with the `csadm SetAuditConfig` command, see [SetAuditConfig](#) (p. 73).



Some events will always and automatically be audited. Other events can be optionally audited if the CryptoServer is configured accordingly.

If an auditable event occurs, the operating system SMOS automatically adds an entry to the audit log file. The audit log entry always includes:

- A timestamp with date and time of the event (if the RTC is running),
- Username of all users who authenticated the audited command (if appropriate),
- Function code (FC) and subfunction code (SFC) of the audited command (if appropriate),
- Error code which indicates if the action has been successfully performed or if an error had occurred (if appropriate).

Additionally, event-individual information may be stored, too. The following events are always logged:

<b>Event</b>	<b>Audit log contains additional information about</b>
New Alarm	Alarm description
The temperature exceeds the valid range	Measured temperature
<code>Clear</code> command performed	
<code>ResetAlarm</code> command performed	The special audit message class is used for this event
<code>SetAuditConfig</code> command performed	Old and new configuration values
FIPS restrictions applied mode	
FIPS mode	
FIPS mode entered	
FIPS mode left	
<code>Clear=DEFAULT</code> command performed	ERASE to factory settings
Erase executed	

Table 2: Events that are always logged into the audit log

Additionally, many more events may be logged if the CryptoServer's audit system is accordingly configured. In the last column it is indicated, if the listed event is audited by a CryptoServer in default configuration.

<b>Auditable event</b>	<b>Audit message class number</b>	<b>Audited by default?</b>
Firmware and file management (e.g., commands to load or delete a file/firmware module, or load the Firmware Encryption Key)	1	yes
User management actions (e.g., commands to add or delete a user or to change the user's authentication token)	2	yes
Setting of the system clock of the CryptoServer	3	yes
CryptoServer startup	4	yes
Audit Log file management (e.g., deletion the audit log file, any changes in the audit configuration)	5	yes
Master Backup Key management (e.g., create or import a Master Backup Key)	6	yes

<b>Auditable event</b>	<b>Audit message class number</b>	<b>Audited by default?</b>
Key management (e.g., key management functions of the firmware module CXI)	7	no
Successful authentications/logins	8	no
Failed authentications/logins	9	yes
Backup and restore operations (e.g., backup database or restore database)	10	yes
Operating system events	11	yes
Action Needed	12	yes
Audit message classes reserved for future use	13 – 23, 31	no
Customer-defined audit message classes	24 - 30	no
Audit message class always generating an audit log entry	-	no
Special audit message class	-	no

Table 3: Auditable events

Additionally to the kind of events that should be logged, it can be configured if the logfiles should be written in rotating order, the maximum number of audit log events and the maximum file size.

Further auditing may be implemented application-dependent. Customer-individual firmware modules can use the audit functionality, for example, for auditing security-relevant actions like key management actions and others more. If this is wanted, the individual firmware module has to call the appropriate CMDs function `cmds_audit_write()`, which writes a message into the CryptoServer's audit log file. Furthermore, the audit log system has to be configured as described above, i.e., by adding the respective audit message classes.



The audit log files are stored in the FLASH memory of the CryptoServer. Regarding the configuration of auditing, it should always be kept in mind that a too extensive auditing may result in a great wear and tear of the FLASH memory and therefore in a decreased lifetime of the CryptoServer. You can verify which audit log files are present in the FLASH memory by performing a command according to the following example:

```
csadm Dev=3001@127.0.0.1 ListFiles | grep .log
```

Example output:

```
FLASH\audit000000.log 747
```

### 3.3.2 Signed Audit Logs

The CryptoServer can generate signed audit log files on the computer csadm is running on.

The audit log signature key, which is described in *Audit Log Signature Key* in the [CryptoServer - Administration Manual \(p. 243\)](#), is used for this process. The generated files have extra long audit log file names, see *AuditLogFileNames* in the [CryptoServer - Administration Manual \(p. 243\)](#). For details about how to generate and use this key, see [Generating and Verifying Signed Audit Log Files \(p. 225\)](#).

### 3.4 Signed Configuration Files

The device software supports the use of signed configuration files with the `.scf` file extension. In such a signed configuration file you can define specific firmware module configuration settings, for example, to disable selected device functions which enables the hardening of these functions, and to define elevated permission/authentication requirements for specific device functions.

The signed configuration files are protected with a signature using an Alternative Module Signature Key, and can only be loaded into the device by a user with system administrator permission (2 in the user group 6). Like this setting, changing or removing additional security rules defined in a signed configuration file requires the same permissions and protection as those for loading and changing the device firmware.

A signed configuration file is originally created as a configuration file with any file extension, for example, `*.cfg`, `*.txt`. This configuration file is then signed with the Module Signature Key or the Alternative Module Signature Key by using the `csadm` command `SignConfig`. The Alternative Module Signature Key is created by the customer outside the device and imported into it (by using the `csadm` commands `GenKey` and `LoadAltMdlSigKey`). The `.scf` file is loaded into the device with the `csadm` command `LoadFile`. While loading it into the device the signature of the `.scf` file is verified by the ADM firmware module. After successful signature verification and before the signed configuration file is stored on the device's file system, the signature is replaced by a SHA-512 hash value. On every startup this hash value is recalculated and compared with the previously stored one, in order to check the integrity of the signed configuration file.

#### Syntax

The syntax of a signed configuration file is as follows:

- Sections are defined as `[Section name]`.
- Global configuration items should be specified above the first section definition.
- Configuration items are defined as combination of a key and a value:  
`<key> = <value_1>,<value_2>,...,<value_n>`



## Rules

- Leading and trailing space characters are stripped on `<keys>` and `<values>`.
- A `<key>` should not contain any space characters.
- The character `#` should be used for commenting out a line.
- Lines with an invalid syntax are ignored.
- One `[Section]` should be used only once in the current configuration file.
- A `<key>` should be used only once in the current `[Section]`.

If there are several `<key>` entries, only the first one applies. The others are ignored.

- The `<value>` might be split over several lines which should end with a backslash `"\"`.

## Deleting and Replacing Signed Configuration Files

Signed configuration files are only deleted when:

- Executing the csadm command `Clear`
- Loading a new firmware package into the device (e.g. by using the csadm command `LoadPkg` with parameter `ForceClear`)
- Executing a Clear-to-Factory-Defaults by performing an External Erase

Signed configuration files cannot be deleted by using the csadm command `DeleteFile`, and are not deleted when an alarm has occurred.

You can load a new signed configuration file into the device with the csadm command `LoadFile`. The existing one is then overwritten.

The signed configuration files are supported for the CryptoServer-Series listed in the table below, and starting with the given minimum versions of the firmware modules making this functionality available:

<b>Minimum versions of required firmware modules</b>	<b>CryptoServer Series</b>	
	<b>CSe</b>	<b>Se Gen2</b>
SMOS	≥ 4.4.0.0	≥ 5.1.0.0
CMDS	≥ 3.2.0.2	≥ 3.2.0.2
ADM	≥ 3.0.12.0	≥ 3.0.12.0

Table 4: Firmware requirements for the support of signed configuration files

### 3.4.1 Sample Configuration File `cmds_sample.cfg`

Utimaco provides a sample configuration file `cmds_sample.cfg` in one of the following directories:

- `\Software\Windows\Administration`  
This directory is to be found on the general SecurityServer bundle.
- `\Software\Linux\Administration`  
This directory is to be found on the general SecurityServer bundle.

It contains a list of the device firmware modules, providing external interfaces, and their unique module IDs (function code (FC)), as well as complete list of their external functions/interfaces (subfunction codes (SFC)). You can use this sample configuration file as a basis for your own configuration/signed configuration file. After editing the sample configuration file to fit to your requirements, you should rename it to `cmds.cfg` so that it can be interpreted by the device.

See also [Creating and Using the Signed Configuration File `cmds.scf`](#) (p. 222).

### 3.4.2 Interface Hardening by Disabling Selected Functions

The device software offers the possibility to additionally secure the device interfaces by disabling selected device functions. These functions have to be defined in the signed configuration file `cmds.scf`.

For that purpose, the configuration file `cmds_sample.cfg` has to contain the dedicated section `[DisableSFC]` specifying which functions should be disabled. The syntax is as follows:

```
[DisableSFC]
<FC> = <SFC1>,<SFC2>,...,<SFCn>
or for better readability
<FC> = <SFC1>,\
      <SFC2>,\
      ..., \
      <SFCn>
```

FC is the function code (ID) of the firmware module, which is exactly three hex digits with leading 0x, e.g., "0x012".

SFC is the specific decimal sub-function code (ID), specifying a function within a firmware module, which is disabled. The SFCs do not have to be ordered numerically.

The disabled firmware module functions are listed during device startup in the Boot Log in specific entries with the syntax

```
CMDS/DSOM: <FC> - disabled <SFC1> <SFC2> ... <SFCn>.
```

For example, `CMDS/DSOM: 0x083 - disabled 0 1 17` means that in the firmware module CMDS (with function code FC = 0x83) the functions Echo (with SFC = 0), Reverse Echo (with SFC = 1) and Set Maximum Authentication Failures (with SFC = 17) are disabled, and cannot be used by external applications.

If you try to access a disabled device function the following error message is returned by the device:

```
Error B0830061
```

This function is not available in this HSM configuration.

See [Creating and Using the Signed Configuration File `cmds.scf`](#) (p. 222) for step-by-step instructions on how to create/edit, sign, load and activate your signed configuration file `cmds.scf`.

### 3.4.3 Configurable Role-based Access

Role-based access provides the possibility to increase the necessary authentication level for selected device functions, and thus, to even further restrict the access to the system to users with customized roles/permissions.

The CryptoServer maximal possible authentication status that can be reached for selected functions in the different user groups has been increased to 15 (0xF).

The increased permission level enabling specific users to execute specific functions can be individually configured by the customer for the device external functions with help of the signed configuration file `cmds.scf`.

In order to do so, the configuration file `cmds_sample.cfg` has to contain a section `[Permissions]` specifying the increased permissions required for specific functions in the defined firmware modules. The syntax is as follows:

```
[Permissions]
<FC> = <SFC1>:<permissions>,<SFC2>:<permissions>,...,<SFCn>:<permissions>
or for better readability
[Permissions]
<FC> = <SFC1>:<permissions>,\
    <SFC2>:<permissions>,\
    ..., \
```

```
<SFCn>:<permissions>
```

FC is the function code (ID) of the firmware module which is exactly three hex digits with leading 0x, e.g., 0x012.

SFC is the decimal sub-function code and permissions is a hexadecimal number (maximum 8 bytes without leading 0x, e.g., FF00F000) denoting the permission in each of the eight user groups. The SFCs do not have to be ordered numerically. The permissions defined in the signed configuration file can only be used to extend the permission required, by default, for the execution of the corresponding firmware module function. For security reasons it is not possible to suspend the required default permissions for the specified functions for external interfaces.

The signed configuration file `cmds.scf` is evaluated during startup, while all firmware modules register with their FC and SFCs at the CMDS module. In case there is an entry for a given firmware module, identified by its FC, found in the `cmds.scf` file, the required permissions for each specified SFC are set according to that definition. The list of customized permissions that are enforced after registration, is included in the audit log only if the log event "Startup messages" is activated, as by default. An entry in the Audit Log contains the following information:

```
<date> <time> <FC:0xXXX> <SFC:XX> Configured Permission=<permission>
```

### For example

```
16.12.2015 13:20:45 FC:0x068 SFC:17 Configured Permission=6F000000
```

During command execution, the currently reached authentication level for the device function, identified by its FC and SFC, is compared to the required permission in the configuration file. If they are equal or no permission restrictions list exists for this module (FC), the command is executed normally performing the default (unchanged) permissions check for the command.

If the required customized permissions are not reached or in case the check of the default permissions fails, the error message B0830001 permission denied is returned.

See [Creating and Using the Signed Configuration File `cmds.scf` \(p. 222\)](#) for step-by-step instructions on how to create/edit, sign, load and activate your signed configuration file `cmds.scf`.

### 3.4.4 List of External Firmware Functions

See the `cmds_sample.cfg` file for a list of the FCs and SFCs of all external firmware functions that can be blocked, and the default permissions required for their execution, which might be increased on demand. This file is available by default in the `<install_dir>\Administration` directory.

## 4 Administration of CryptoServer with csadm

If the CryptoServer is in error state, no cryptographic services are available. Only the following commands are available:

- `csadm GetState`
- `csadm GetTime`
- `csadm ShowAuthState`
- `csadm GetBootLog`
- `csadm GetAuditLog`
- `csadm ListFiles`
- `csadm ListFirmware`
- `csadm Clear=DEFAULT`

### 4.1 Overview of csadm

#### 4.1.1 Syntax of csadm

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

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

Please note:

- Parameters and commands are processed from left to right.
- If a subsequent command requires to set a parameter or to run another command prior to its own execution, it will have to be entered rightmost.
- Expressions in squared brackets [] are optional.
- In the syntax describing line of the individual command descriptions all parameters are shown in angle brackets < > and are explained later.
- Some parameters or commands require an assigned value ('=...'), some do not.

- Some commands use a default value if none is given ("[-...]").



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

The following is an example of a correct csadm command entry in the Microsoft PowerShell:

```
csadm [Dev=<device>] LogonSign="<user>,<keyspec>" <command>
```



The csadm tool does not support POSIX syntax. Therefore, the use of "~" is not supported. Only relative and absolute path may be specified.

Examples:

```
csadm dev=PCI:0 GetState
```

```
csadm dev=TCP:192.168.1.98
... LogonSign=ADMIN,"C:\Program
Files\Utimaco\SecurityServer\Administration\ADMIN.key" ...
... LoadFile="C:\Program Files\Utimaco\SecurityServer\Firmware\exmp.mtc"
```

```
csadm dev=192.168.1.1 LogonPass=sven,ask
... AddUser=nils,00000002,hmacpwd,ask
```

```
csadm dev=TCP:288@154.54.2.23 LogonSign=ADMIN,:cs2:cjo:USB0
... SetTime=20020602111532
```

Every command running on CryptoServer or CryptoServer LAN requires the device parameter Dev= which sets the address of the CryptoServer or CryptoServer LAN. Commands running locally without using a CryptoServer do not need this parameter. Possible values are:

Device address	Description
/dev/cs2.n where n = {0, 1, 2, ..., 7}	Local CryptoServer No. n+1 on a UNIX system. The maximum number of eight CryptoServer PCIe cards can be changed in the source of the Linux driver.

<b>Device address</b>	<b>Description</b>
<code>PCI : n</code> where $n = \{0, 1, 2, \dots, 31\}$	Local CryptoServer No. $n+1$ on a Windows system
<code>TCP : 288@194.168.4.107</code>	IP address and port number of a CryptoServer LAN In csadm commands, always use IP addresses without leading zeros although they are shown on the CryptoServer LAN display, e.g., <code>194.168.004.107</code> .
<code>TCP : 194.168.4.107</code>	IP address of a CryptoServer LAN (default: port=288) In csadm commands, always use IP addresses without leading zeros although they are shown on the CryptoServer LAN display, e.g., <code>194.168.004.107</code> .
<code>194.168.4.107</code>	IP address of a CryptoServer LAN (default: protocol=TCP, port=288) In csadm commands, always use IP addresses without leading zeros although they are shown on the CryptoServer LAN display, e.g., <code>194.168.004.107</code> .
<code>TCP : 288@cslan01</code>	Host name and port number of a CryptoServer LAN (using DNS request to resolve host name)
<code>TCP : cslan01</code>	Host name of a CryptoServer LAN (using DNS request to resolve host name, default: port=288)
<code>cslan01</code>	Host name of a CryptoServer LAN (using DNS request to resolve host name, default: protocol=TCP, port=288)
<code>TCP : 3001@127.0.0.1</code> or <code>TCP : 3001@localhost</code>	Local CryptoServer Simulator for Windows/Linux (SDK)
<code>3001@127.0.0.1</code> or <code>3001@localhost</code>	Local CryptoServer Simulator for Windows/Linux (SDK) with the default protocol TCP

Table 5: Device Addresses



If the CRYPTOSERVER environment variable is set according to the above-mentioned syntax, the Dev= parameter can be skipped.

Using the `Dev=` parameter overrides the CRYPTOSERVER environment variable in any case for the specific command in any case.



### 4.1.2 Command Execution with the csadm Tool

If the CryptoServer is mounted in the local computer (as a PCIe card) commands are sent from the host to the CryptoServer via the PCIe interface. The CryptoServer processes the command and sends the answer (answer data or error code) back to the host.

If the CryptoServer is part of a CryptoServer LAN, commands are sent from the host to the CryptoServer via a TCP connection. Generally, the TCP server (daemon) running on the CryptoServer LAN (csxlan) forwards incoming commands to the mounted CryptoServer, but a few commands are responded by the CryptoServer LAN itself.

The following figure shows how commands are executed on a local CryptoServer PCIe card or on a remote CryptoServer LAN:

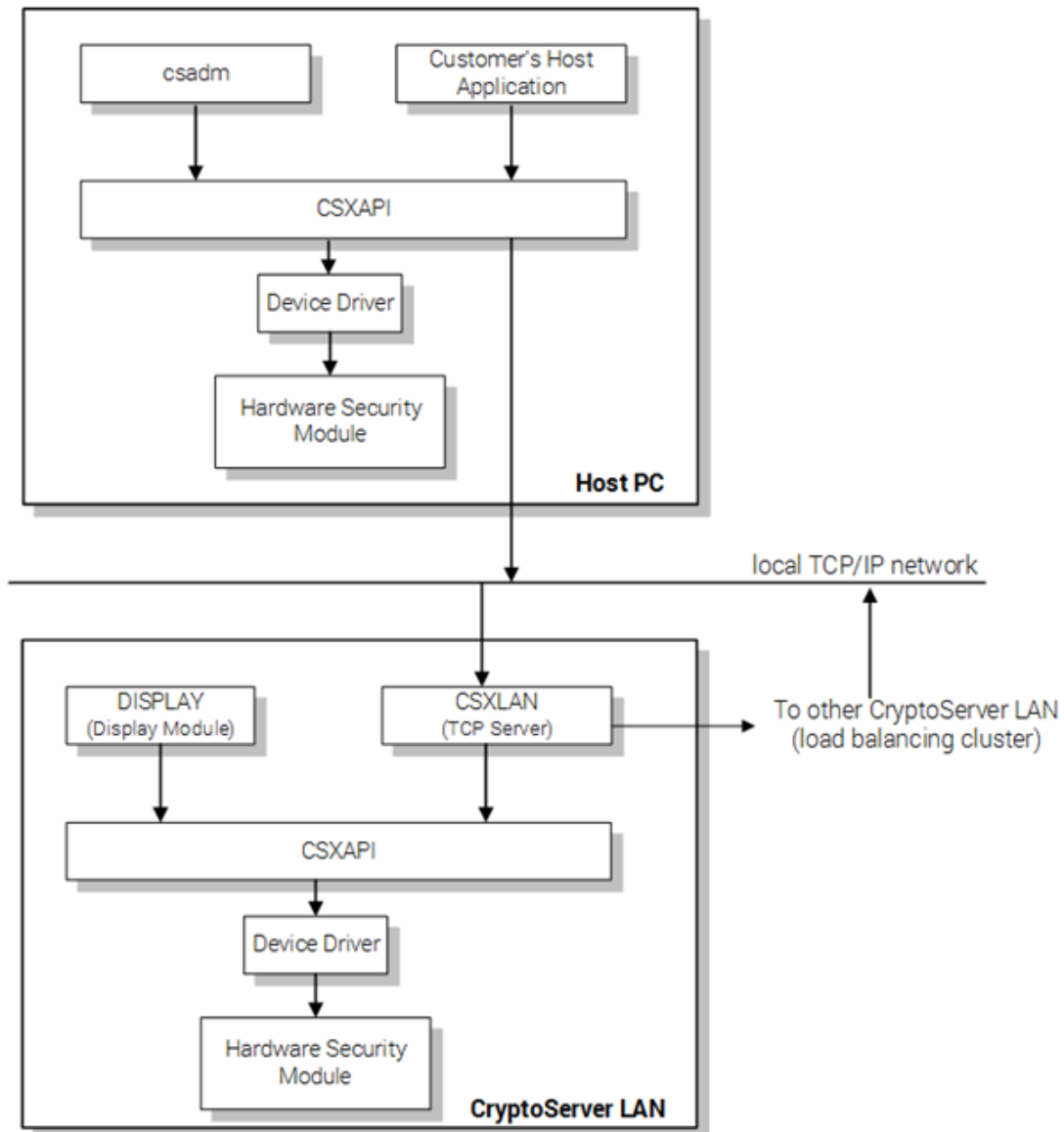


Figure 1 : csadm Command Execution

An application on the host computer can use the command execution library CSXAPI to execute any command on a CryptoServer PCIe card or CryptoServer LAN.

CSXAPI (CryptoServer Extended Application Programming Interface) is a software running on the host that has the task to generate a C-interface out of the external CryptoServer byte stream interface.

As a standard application the CryptoServer Administration Tool csadm provides any kind of basic administration like file loading or deletion, setting the CryptoServer's clock, user management, etc.



From the user's point of view, it makes no difference whether he/she accesses a local CryptoServer PCIe card or a remote CryptoServer LAN. The CryptoServer Administration Tool csadm is able to send commands either to the local CryptoServer or to the remote CryptoServer LAN (by choosing the appropriate device address).

Commands can be divided into the following groups:

<b>Command Destination</b>	<b>Counterpart on the CryptoServer</b>	<b>Command Group</b>
CryptoServer	Command Scheduler Module (CMDS)	authentication, user management
	Administration Module (ADM)	administration of CryptoServer card
	MBK Management Module (MBK)	management of Master Backup Keys
	other firmware modules	project specific (not realized in csadm)
TCP Server of the CryptoServer LAN	Control module of the TCP Server (daemon) csxlan	administration (configuration) of the TCP Server ('daemon') csxlan

Table 6: csadm command groups

### 4.1.3 Password Authentication

To authenticate a security-relevant administration command, an operator who uses a password-based authentication mechanism like HMAC Password Authentication, see *Authentication Mechanisms* in the [CryptoServer - Administration Manual \(p. 243\)](#), has to enter their username and password to the csadm tool, see [Commands for Authentication \(p. 46\)](#). At this and other opportunities it is possible to read the password on the monitor which is connected to the PC on which csadm is running.



To avoid the password being reflected as plaintext on the monitor, csadm 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 prompt for the password separately, before starting to process the authentication (and the rest of the command).

Example:

```
csadm LogonPass=sven,ask SetTime=GMT  
Enter Passphrase:
```

If now the password is entered over the keyboard, it is not reflected in clear text on the monitor, but is replaced by default characters on the display.

The hidden password entry mechanism can also be used to protect the password of an encrypted keyfile, see [Command Execution with the csadm Tool \(p. 33\)](#) or the root password of the CryptoServer LAN from being displayed on the monitor.



We strongly recommend using a hidden password entry.

---

#### 4.1.4 Storage and Specification of RSA and ECDSA Keys for Authentication

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

- RSA/ECDSA keys stored in a keyfile `*.key` (as plaintext)
- RSA/ECDSA keys stored in an encrypted keyfile `*.key` (private key part stored encrypted, public key part stored as plaintext)
- RSA/ECDSA keys stored on a smartcard.

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

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

Encrypted RSA keyfiles are protected by a 168-bit Triple-DES key that is derived from a password with the SHA-256 hashing algorithm. Encrypted ECDSA keyfiles are protected by a 256 bit AES key that is derived from a password with the SHA-256 hashing algorithm. In both cases the password can be changed with the `csadm ChangePassword` command. With

the same command a plaintext keyfile can be changed in an encrypted keyfile and vice versa (by omitting the old respectively the new password).



Apart from test environments, we strongly recommend storing private keys on smartcards. Only in this case the private key will never leave the secure token and not be used for calculations on the host.



For all commands that use RSA or ECDSA keys a key specifier is required in the command syntax. A key specifier is either a name of a keyfile ( `*.key` ) or a smartcard specifier.

#### 4.1.4.1 Key Specifiers

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

Example1:

```
csadm LogonSign=ADMIN,c:\my_keys\myKey.key#sIlEnCe DeleteUser=sven
```

Example 2:

```
csadm LogonSign=ADMIN,c:\my_keys\myKey.key#ask DeleteUser=sven
```

If no password is given (or the password is replaced by the string 'ask'), hidden password entry will be performed for encrypted keyfiles, which we strongly recommend.

#### Key Specifier Examples

Key Specifier	Description
C:\my_keys\myKey.key	Keyfile
C:\my_keys\myKey.key#mypassword	Keyfile including a password
C:\my_keys\myKey.key#ask	Keyfile including an interactive password retrieval



If RSA signature authentication with a smartcard is used and if the PIN pad is not connected to a different computer than the one csadm is running on, see *Using a Local PIN Pad for a Remote CryptoServer* in the [CryptoServer - Administration Manual \(p. 243\)](#).

For details and examples, see [LogonSign \(p. 48\)](#).

#### 4.1.4.2 Smartcard Specifiers

A smartcard specifier always starts with a colon and consists of three strings separated by colons (for example, `:cs2:cjo:USB0`):

- The first string identifies the type of the smartcard.
- The second string identifies the type of the smartcard reader (PIN pad).
- The last string is the name of the serial device or the USB device the reader is connected to.

Currently, only the smartcards of type TC30 and JavaCard, with the identifier cs2, are supported:

The following types of smartcard readers (PIN pads) are supported:

<b>PIN Pad Identifier</b>	<b>Smartcard reader type (PIN pad types)</b>
cyb	REINER SCT cyberJack (COM) or REINER SCT cyberJack (USB)
cjo	Utimaco cyberJack one
auto	Automatic detection of the PIN pad type

Table 7: Supported PIN pads

The following port types are supported:

<b>Port Identifier</b>	<b>Description</b>
USBn where n = {0, 1, ...}	USB port No. n+1
USBn[@<port>]@<IP address>[/<password>] [#<smartcard PIN>] where n = {0, 1, ...}	For local PIN pad and remote CryptoServer Tools/ API only

Table 8: Supported ports

In addition to that, a smartcard PIN preceded by a # can be appended. Use this option, if you do not want to enter the PIN at the PIN pad.



The PIN you enter here is shown in plaintext in your command line. There is no option to hide it as it can be done for example for a password in the `csadm LogonSign` or the `csadm LogonPass` command.

If the PIN pad is used without using the PIN pad daemon, the smartcard PIN may even be used when a PIN is not expected. Otherwise, an error message is shown.

Example: `csadm GetCardInfo=:cs2:cjo:USB0#123456`

### Key Specifier Examples

<b>Key Specifier</b>	<b>Description</b>
<code>C:\my_keys\myKey.key</code>	Keyfile
<code>C:\my_keys\myKey.key#mypassword</code>	Keyfile including a password
<code>C:\my_keys\myKey.key#ask</code>	Keyfile including an interactive password retrieval
<code>:cs2:cjo:USBn</code> or <code>:cs2:auto:USBn</code> where <code>n = {0, 1, 2, ...}</code>	Key from a smartcard using a Utimaco cyberJack one connected to a USB port of a Windows or Linux computer

Table 9: Examples for key specifiers

If RSA or ECDSA signature authentication with a smartcard is used and if the PIN pad is not connected to a different computer than the one csadm is running on, follow the instructions in *Using a Local PIN Pad for a Remote CryptoServer* in the [CryptoServer - Administration Manual](#) (p. 243).

## 4.2 Basic Commands

These basic commands are csadm internal functions.

For their execution no connection to the CryptoServer is established.

### 4.2.1 Help

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

<b>Syntax</b>	csadm Help csadm Help=<command>
---------------	------------------------------------

<b>Parameter</b>	<b>Description</b>
<command>	Specific csadm command to display help for

<b>Example</b>	csadm Help=ListFiles
----------------	----------------------

<b>Output</b>	Upon successful execution, the command returns a list of all csadm command, or specific information on command and parameters if a csadm command was specified.
---------------	---

### 4.2.2 More

This command shows a list of the csadm commands which are used for the load preparation of firmware modules.

<b>Syntax</b>	csadm More
---------------	------------

<b>Parameter</b>	<b>Description</b>
No parameters are given for input.	

<b>Example</b>	csadm More
----------------	------------

<b>Output</b>	Upon successful execution of the command, the list of all rarely used csadm commands for firmware packaging, mostly used by SDK customers who create individual firmware modules, is returned.
---------------	--



### 4.2.3 PrintError

This command displays the corresponding error message text to an error code. csadm has a built-in list with all standard error messages of the device and Host-APIs.

<b>Syntax</b>	csadm PrintError=<errorcode>
<b>Parameter</b>	<b>Description</b>
<errorcode>	Error code (hexadecimal)
<b>Example</b>	csadm PrintError=B901306F
<b>Output</b>	Upon successful execution, the command returns a list of all csadm command, or specific information on command and parameters if a csadm command was specified. Error B901306F CryptoServer API LINUX can't get connection errno = 11

### 4.2.4 StrError

This command displays the corresponding error message text to an error number (which might be system specific).

<b>Syntax</b>	csadm StrError=<errornumber>
<b>Parameter</b>	<b>Description</b>
<errornumber>	Error number
<b>Example</b>	csadm StrError=7
<b>Output</b>	Upon successful execution of the command, the error message text to given error number is returned.

### 4.2.5 Version

This command shows the version numbers of the csadm tool and the included libraries.

<b>Syntax</b>	csadm Version
---------------	---------------

<b>Parameter</b>	<b>Description</b>
No parameters are given.	

<b>Example</b>	csadm Version
----------------	---------------

<b>Output</b>	<p>Upon successful execution, the command returns the version of the installed csadm tool and the included libraries.</p> <pre>csadm 2.4.2 csadm_lib (global) 3.5.0 csapi 1.11.1 (Dec 5 2018) pp_api 1.9.4 (Dec 5 2018) sl 1.1.3 yac1 1.13.0</pre>
---------------	--

### 4.2.6 SetTimeout

With this command the maximum time that the driver waits for a response from the device can be changed for the subsequent commands.



The new timeout is valid for the current connection. Next time csadm is executed, the default timeout=600000 ms will be used.

<b>Syntax</b>	csadm [Dev=<device>] SetTimeout=<timeout> <commands>
---------------	--

<b>Parameter</b>	<b>Description</b>
<device>	<p>Device address</p> <p>This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.</p>

Parameter	Description
<timeout>	New timeout to be set in milliseconds
<commands>	Subsequent commands which should be executed with the new timeout

<b>Example</b>	<code>csadm Dev=192.168.1.1 SetTimeout=10000 GetState</code>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the current settings of default timeout and chosen timeout are returned.</p> <pre>default timeout: 600000 ms chosen timeout: 10000 ms</pre>
---------------	--

## 4.2.7 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 administration computer.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of the `csadm Cmd` command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

An example for the Microsoft PowerShell:

```
csadm [Dev=<device>] LogonSign="<user>,<keyspec>"
Cmd="0x87,0x05,1,2,3"
```



Although in theory it is possible to send commands of up to 256 kB to the device 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..

<b>Syntax</b>	<pre>csadm [Dev=&lt;device&gt;] [&lt;Authentication&gt;] ... Cmd=&lt;fc&gt;,&lt;sfc&gt;,&lt;byte_n&gt;,...</pre>
---------------	--

<b>Authentication</b>	Depending on specified command
-----------------------	--------------------------------

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<fc>	Function code, module ID of the firmware module Can be either in hexadecimal format 0x00, ..., 0xFF or in decimal format 0, ..., 1023
<sfc>	Sub function code, number of the called function Can be either in hexadecimal format 0x00, ..., 0xFF, or in decimal format 0, ..., 255
<byte_n>	n <sup>th</sup> data byte, dependent on the command Can be either in hexadecimal format 0x00, ..., 0xFF or in decimal format 0, ..., 255

<b>Example</b>	csadm Dev=192.168.1.1 LogonPass=sven,ask Cmd=0x123,0x05,1,2,3,4,5,6,7,8
----------------	--

<b>Output</b>	Upon successful execution of the command, the generic command interface of the specified command is given.
---------------	--

## 4.2.8 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:

<b>Character</b>	<b>Name</b>	<b>Description</b>
#	hash sign	rest of line is comment
,	comma	separator
	space	separator
<b>TAB</b>	tabulator	separator
<b>CRLF</b>	new line	separator
<b>hex</b>	tag	interpret all values as hexadecimal from now on, even if '0x' is omitted
<b>dec</b>	tag	return to normal mode (i.e., hexadecimal interpretation requires a leading '0x')
<b>"..."</b>	string	string which can reach the end of the line

Table 10: Characters and strings allowed in a command file

**Example**

```
#
# demo command file
#
0x123  # function code / module ID
0x05   # sub function code
0x00,0,0x00,11 # hexadecimal and decimal notation may be mixed
"Hello World"  # strings are framed with quotation marks
hex 0F,1E,2D,3C,4B,5A,60,59 # leading hex-tag allows omitting of '0x'!
68,77,86,95,A4,B3,C2,D1 # still understood as hexadecimal values
dec # return to normal notation
11,22,33   # decimal values
```

**Syntax**

```
csadm [Dev=<device>] [<Authentication>] CmdFile=<file>
```

**Authentication**

Depending on specified command

<i>Parameter</i>	<i>Description</i>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<file>	Name and extension of the file do not matter

**Example**

```
csadm LogonPass=sven,swordfish CmdFile=demo.cmd
```

**Output**

Upon successful execution of the command, the generic command interface of the specified command is given.

## 4.2.9 Sleep

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

**Syntax**

```
csadm Sleep=<time> <commands>
```

<b>Parameter</b>	<b>Description</b>
<time>	Time delay in seconds
<commands>	Further command whose execution should be delayed by the given time

<b>Example</b>	csadm StartOS Sleep=3 GetState
----------------	--------------------------------

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

### 4.2.10 ConnTimeout

With this command the device connection timeout can be changed for the subsequent commands. The new timeout is valid for the current connection. Next time csadm is executed, it will use the default connection timeout setting.

<b>Syntax</b>	csadm [Dev=<device>] ConnTimeout=<timeout> <commands>
---------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<timeout>	New connection timeout to be set in milliseconds
<commands>	Subsequent commands which should be executed with the new connection timeout

<b>Example</b>	csadm Dev=4001@194.168.4.107 conntimeout=10000 GetState
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

## 4.3 Commands for Authentication

The device provides a variety of command authentication mechanisms.

Most pre-defined security-relevant commands require the authentication level 2 in a specific user group. Since each user usually has only permission 1 in one or more user groups (apart

from the pre-defined user ADMIN who has permission 2 in the user groups 7 and 6, i.e., for the user management and for the administrative commands), this means that two users of the specific user group are necessary to authenticate the command. For this reason, the specific command requires authentication according to the two-person rule, i.e., authentication by two independent users is required for the command to be executed.



If a command requires multi-user authentication according to the n-person rule ( $n=1\dots 16$ ), all  $n$  users have to apply their authentication prior to command execution:

```
csadm <Authentication_1> <Authentication_2> ... <Authentication_n> <command>
```

The users may even use different authentication mechanisms.

Example:

```
csadm LogonSign=roberta.:cs2:cjo:USB0 LogonPass=sven,ask  
DeleteFile= mdlsigalt.key
```

Consider that the authentication statuses can be added without any further restriction only if the involved users are logged in to perform a command:

- of a firmware module that is not the CXI firmware module or
- of the CXI firmware module and the cryptographic key (CXI key) that is used to perform this command does not belong to any key group.

If however, the used cryptographic key in the CXI firmware module is assigned to a key group, only the authentication statuses of those logged in users can be summed up that belong to the same key group.

Example:

userA has authentication status 0x00000001 and is assigned to the key group A.

userB has authentication status 0x00000001 and is assigned to the key group B.

Three cryptographic keys are created. keyA is assigned to key group A, keyB to key group B and keyAB to key group AB.

If userA and userB are simultaneously logged in, only those commands of the CXI firmware module can be performed that only need

- an authentication status 0x00000001 for using keyA or
- an authentication status 0x00000001 for using keyB.

If a command needs

- an authentication status 0x00000001 or higher for using keyAB or
- an authentication status 0x00000002 or higher for using keyA or
- an authentication status 0x00000002 or higher for using keyB,

it cannot be performed.



The authentication commands LogonSign and LogonPass do not only perform command authentication, but provide also a Secure Messaging session for the protection of the confidentiality of the command data.

These authentication commands (leading to the necessary authentication status) can be inserted for the placeholder `<Authentication>` which are given in the syntax of all security relevant commands described in the current chapter.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary

### 4.3.1 LogonSign

With this command a user with signature-based authentication mechanism (i.e., authentication mechanism RSA Signature, ECDSA Signature or RSA Smartcard) opens an authenticated Secure Messaging session for the given command. The session key (32-byte AES) is generated by using one of the following ways:

- With the Ephemeral Unified Model Key Agreement Scheme according to SP800-56A r3 (FIPS-compliant; based on EC Diffie Hellman)

This is automatically tried first. If the device does not support this, the Diffie-Hellman key establishment protocol is used.

- With the Diffie-Hellman key establishment protocol





The authenticated Secure Messaging session is automatically closed after the command execution and csadm ends.

For any further csadm command which needs authentication and/or confidentiality, a new authenticated Secure Messaging session has to be opened.

### Syntax

```
csadm [Dev=<device>] ... LogonSign=<user>,<keyspec> [<further
Authentication>] <command>
```

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<user>	Username
<keyspec>	<p>Key specifier where the private part of the user authentication key should be loaded from (key specifier of smartcard or keyfile)</p> <ul style="list-style-type: none"> <li>Smartcard specifier: for example, <code>:cs2:cjo:USB0</code> If you append # and a PIN to the smartcard specifier, the user is automatically logged in without any user interaction at the PIN pad. Example: <code>:cs2:cjo:USB0#123456</code> For RSA smartcard authentication, the following applies: <ul style="list-style-type: none"> <li>The smartcard specifier can be omitted. The comma after the user name can be omitted as well (recommended).</li> <li>For an automatic login, replace the smartcard specifier by <code>#&lt;PIN&gt;</code>.</li> </ul> </li> <li>Storage location and name of the keyfile ( <code>*.key</code> ) In case of an encrypted keyfile: <code>*.key[#&lt;password&gt;]</code> where <code>&lt;password&gt;</code> is the password used for protecting the keyfile. It might be entered: <ul style="list-style-type: none"> <li>in clear text, for example, <code>MyKey.key#mypwd</code> or</li> <li>as string ask for hidden password entry, for example, <code>MyKey.key#ask.</code></li> </ul> </li> </ul>
<command>	Command that has to be authenticated

<b>Example</b>	<p>Example 1:  <code>csadm LogonSign=sven,:cs2:cjo:USB0 DeleteFile=example.mtc</code></p> <p>Example 2:  <code>csadm LogonSign=sven,E:\myKeys\myRSA.key#ask LogonPass=nils,ask DeleteFile=example.mtc</code></p> <p>Example 3:  The user administrator AdminSc (22000000) uses RSA smartcard authentication and a Utimaco cyberJack one PIN pad directly connected to the PCIe card. He authenticates a <code>csadm AddUser</code> command to create a user <code>UsrRsaSc</code> (00000002) who should be assigned to the 1234 cryptographic key group, uses RSA smartcard authentication and whose private key should be available on a smartcard inserted in a Utimaco cyberJack one PIN pad connected to the computer <code>csadm</code> is running on.</p> <code>csadm Dev=PCI:0 LogonSign=AdminSc,:cs2:cjo:USB0 AddUser=UsrRsaSc,00000002{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0</code> <p>The following alternatives are possible because <code>AdminSc</code> uses RSA smartcard authentication:  Shorter (no smartcard specifier and a shortened permission) but with the same effect:  <code>csadm Dev=PCI:0 LogonSign=AdminSc, AddUser=UsrRsaSc,2{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0</code></p> <p>Also possible (no comma after the username):  <code>csadm Dev=PCI:0 LogonSign=AdminSc AddUser=UsrRsaSc,2{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0</code></p> <p>With automatic login using the PIN 123456:  <code>csadm Dev=PCI:0 LogonSign=AdminSc,#123456 AddUser=UsrRsaSc,2{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0</code></p>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

If the user uses the RSA signature authentication or the ECDSA signature authentication and if the private part of the user authentication key is stored on a smartcard, the user is prompted at the PIN pad to insert his smartcard and enter the PIN. The PIN pad can be connected to a USB port of the computer where the `csadm` tool is running or to another computer.

A user with RSA smartcard authentication is prompted at the PIN pad to insert her/his smartcard and enter the PIN. The PIN pad has to be connected directly to the USB port of the PCIe card. It is not sufficient to connect the PIN pad to a port of the computer `csadm` is running on. If the user is used on a LAN device, the PIN pad may also be connected to a port on the front panel that is directly connected to the PCIe card. Depending on the LAN device version, this is a USB port labeled **CS USB**, **USB CS**, **USB CS2** or **HSM**. No remote connection

between the PIN pad and the CryptoServer is supported. The instructions for using a Local PIN Pad for a remote device do not apply here.

However, if you use the device Simulator, the smartcard must be inserted in any case (RSA signature authentication, ECDSA signature authentication or RSA smartcard authentication) into a PIN pad that is connected to the computer where the csadm tool is running (USB port) or to another computer.

When administering the device (for example, by performing the csadm `LogonSign` command), the following error message might be shown:

```
Error B9000405
CryptoServer API
authentication layer for CryptoServer
error in signature function
```

In this case, update the PIN pad driver.

The default administrator is the ADMIN user (22000000). His/her default keyfile is the `ADMIN.key` file (default path: `C:\Program Files\Utimaco\SecurityServer\Administration`). In the delivery state of the CryptoServer, this keyfile cannot be used to perform commands the ADMIN user needs an authentication for, except for the `csadm ChangeUser` command. The `csadm ListUser` command shows the `I[1]` attribute value for this user. To enable the ADMIN user to perform commands he/she needs an authentication for, the ADMIN user must change the credentials by performing the `csadm ChangeUser` command. Then the `csadm ListUser` command shows the `I[0]` attribute value for this user.

The only exception of this behavior of ADMIN users is the ADMIN user on a CryptoServer Simulator. He/she uses the `ADMIN_SIM.key` keyfile (default path: `C:\Program Files\Utimaco\SecurityServer\Administration`). The `csadm ListUser` command shows the `I[0]` attribute value for this user. This `ADMIN_SIM.key` keyfile can be used to perform any command the ADMIN user needs an authentication for. We highly recommend to change the credentials of this ADMIN user, but it is not mandatory.

### 4.3.2 LogonPass

With this command a user with password-based authentication mechanism (i.e., authentication mechanism HMAC Password) opens an authenticated Secure Messaging session for the given command. The session key (32-byte AES session key) is generated using one of the following ways:

- With the Ephemeral Unified Model Key Agreement Scheme according to SP800-56A r3 (FIPS-compliant; based on EC Diffie Hellman)

This is automatically tried first. If the device does not support this, the Diffie-Hellman key establishment protocol is used.

- With the Diffie-Hellman key establishment protocol



The authenticated Secure Messaging session is automatically closed after command execution and/or when csadm is closed.

For any further csadm command that needs authentication and/or confidentiality, a new authenticated Secure Messaging session has to be opened.

A user uses an HMAC password-based key-derived function (HMAC-PBKDF) for authentication. The HMAC-PBKDF according to NIST SP 800-132 does not use the HMAC password itself for authentication but a function derived from the HMAC password. For HMAC-PBKDF, 1000 iterations are used here. This number of iterations, as used for the key derivation from a given password, is fix and not configurable. HMAC-PBKDF is only applied if on both sides, the host side and the firmware side, HMAC-PBKDF is applied. If it is not available on one of these sides, the legacy version of the HMAC password-based mechanism is applied, whereby the user's password is used directly as an HMAC key. In FIPS mode, using HMAC-PBKDF is mandatory.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] LogonPass=&lt;user&gt;,&lt;password&gt; ... [&lt;further Authentication&gt;] &lt;command&gt;</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<b>&lt;device&gt;</b>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>&lt;user&gt;</b>	Username
<b>&lt;password&gt;</b>	User password <ul style="list-style-type: none"> <li>▪ String ask if hidden password entry should be used. We strongly recommend using a hidden password entry. In this case csadm requests password input (Enter Passphrase:) and does not echo the input.</li> <li>▪ User password entry in plaintext</li> </ul>
<b>&lt;command&gt;</b>	Command that has to be authenticated

<b>Example</b>	<pre>csadm LogonPass=sven,swordfish DeleteFile=example.mtc</pre> <pre>csadm LogonPass=sven,ask LogonPass=nils,ask DeleteFile=example.mtc</pre>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

Once a user with HMAC password authentication (or more precise: HMAC password-based key-derived function (HMAC-PBKDF) for authentication) has been created, this user cannot perform commands this user needs an authentication for, except for the `csadm ChangeUser` command. The `csadm ListUser` command shows the `I[1]` attribute value for this user. To enable this user to perform commands he/she needs an authentication for, he/she must change the credentials by performing the `csadm ChangeUser` command. It is important that the changes are performed by the user himself/herself and not, for example, by an administrator. Then the `csadm ListUser` command shows the `I[0]` attribute value for this user.

### 4.3.3 ShowAuthState

This command displays the current authentication status on the device and a list of the users who are currently logged on to the device.

The authentication status is displayed as the sum of permissions of all users, who are currently logged on to the device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; ShowAuthState</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<ul style="list-style-type: none"> <li><code>csadm Dev=192.168.1.1 LogonSign=Adm1,:cs2:cjo:USB0 LogonPass=Adm2,ask ShowAuthState</code></li> </ul>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the current authentication state of the device is returned.</p> <pre>current AUTH state: 23000000</pre> <pre>user: Adm1, Adm2</pre>
---------------	--

### 4.3.4 GetHSMAuthKey



This command has been first introduced with the csadm tool provided on the SecurityServer product CD 4.10. The use of the csadm `GetHSMAuthKey` command requires that at least version 3.5.3.x of the firmware module CMDS has been loaded on the device and successfully initialized.

The csadm `GetHSMAuthKey` command retrieves the public part of the HSM Authentication Key which is used for the establishment of a mutually authenticated Secure Messaging session for the communication between the device and the host applications.

For enabling the mutual authentication, some additional steps are required. For example, the output of the csadm `GetHSMAuthKey` command must be copied into a .key file and the CS\_AUTH\_KEYS system environment variable must be set.

<b>Syntax</b>	csadm [Dev=<device>] GetHSMAuthKey
---------------	------------------------------------

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	csadm Dev=10.1.7.1 GetHSMAuthKey
----------------	----------------------------------

<b>Output</b>	<p>Upon successful execution of the command the following output is given:</p> <p>&lt;Serial number of the PCIe card&gt;=&lt;public part of the HSM Authentication Key mixed with other device internal data&gt;</p> <p>The serial number of the PCIe card is in format CS&lt;XXXXXX&gt;.</p> <pre>CS505156=AC5D8198246C4FF407F7E11B4E53C4AFF326BDF350FF531CB26E0A250 DA7CD5C0AE06C69CA8846D75102B80C3D59E06C6CC98130D27EA260CB0836A486E BE5A29A288EDFFCA4307A6F3A9EF0549D343FDAB57012502050BC40AF8A71F4191 4402DE917427D976DB719802BFECA96FCCABEF32A7EF79296A4B5C79D30FC545A6 C7ACE585B48890E243BC9A664E58E9992969DA1A234581083A18C62DB9FEB3B05B B29E4E68C282220DF3939120F125AEC9F6D9533C8A9BC3C890433B218444C81A13 79A671DEDF92273E61282B345E23ECE82AF964E96BD4681678A59077C1CE5F17A5 8D620D72FE845FB68D249BA7E25D924A98269E050D57F4558A29B6BCDE07C30D69 14BF6A06DF5FE9014C8AF3BE1040A96B36D3583976D180520DAF994E3A0A9C9D5D EE836F1ACD606A3B8FAD186ED6AC0FC34C974FCCB5194F87EB3971F8A8DF728B02 ECC2F21ECA4BCB71748435540EF92EDE0A69CA74E070D38857661D63538C64C416 9F6EEE972D0B455734F40435B7B927BFF8982F5AD3C998D304D</pre> <p>In case of the device Simulator, the serial number has the format <code>SI&lt;xxxxxx&gt;</code>.  <code>SI</code> stands for "simulator" and <code>&lt;xxxxxx&gt;</code> stands for the port number the simulator is listening on, i.e., 003001 for the first simulator instance, 003003 for the second simulator instance etc.  Example: <code>SI003001</code></p> <p>In case of the PaymentServer Simulator, the serial number is always <code>CS0000000</code>.</p>
---------------	---

If an alarm has occurred on the device, the execution of the csadm `GetHSMAuthKey` command fails with an error message.

If the Java-based GUI CryptoServer Administration Tool (CAT) starts with the invalid auth. key error message, either delete the CS\_AUTH\_KEYS environment variable or update it by performing the csadm `GetHSMAuthKey` command.

## 4.4 Commands for Administration

In this chapter the commands for the administration of the device are described, like status requests, alarm treatment and audit management.

The device has to be either in Operational Mode or in Maintenance Mode.

Some of the commands have to be authenticated, some do not.

Command authentication is based on a sophisticated user management concept, which allows the creation of various users with different permissions, authentication mechanisms and other properties.

Although command authentication 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, the description of the command syntax for commands in Operational Mode, which have to be authenticated, does not specify each possibility of authentication. Instead, a placeholder (`<Authentication>`) is inserted, and the command example shows one possibility of authenticating the command.



If the syntax of one of the commands described in the following chapters contains the placeholder `<Authentication>`, it has to be replaced by one or more authentication commands according to the required authentication status and depending on the specific user's permissions and authentication mechanism.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

#### 4.4.1 GetState

This command returns the status of the device, its temperature, alarm state, hardware information and set-up information.



The csadm `GetState` command is responded by the device in any mode (Bootloader, Maintenance Mode as well as Operational Mode) and therefore should be executed as first diagnostic measure in case of problems.

Please prepare the output of `GetState` in case of a support request.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetState</code>
---------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=10.1.7.1 GetState</code>
----------------	--



<b>Output</b>	<p>Upon successful execution of the command, the information on the state of the device is returned.</p> <p>Example 1:</p> <pre> mode      = Operational Mode - Administration-Only state     = INITIALIZED (0x00100004) temp      = 39.8 [C] alarm     = OFF bl_ver    = 4.00.1.3           (Model: CSe-Series) hw_ver    = 4.00.3.0 uid       = b0000011 0c310101   1 adm1      = 43536531 30202020 43533539 30303032   CSe10 CS590002 adm2      = 5554494d 41434f20 43533539 30303032   UTIMACO CS590002 adm3      = 494e5354 414c4c45 44000000 00000000   INSTALLED </pre> <p>Example 2:</p> <pre> mode      = Operational Mode state     = INITIALIZED (0x08100004) temp      = 39.8 [C] alarm     = OFF bl_ver    = 4.00.1.3           (Model: CSe-Series) hw_ver    = 4.00.3.0 uid       = b0000011 0c310101   1 adm1      = 43536531 30202020 43533539 30303032   CSe10 CS590002 adm2      = 5554494d 41434f20 43533539 30303032   UTIMACO CS590002 adm3      = 494e5354 414c4c45 44000000 00000000   INSTALLED </pre> <p>Example 3:</p> <pre> mode      = Maintenance Mode state     = INITIALIZED (0x000a7f84) temp      = 39.8 [C] alarm     = ON sens      = 027f            - Alarm has occurred            - external Erase is executed  bl_ver    = 4.00.1.3           (Model: CSe-Series) hw_ver    = 4.00.3.0 uid       = b0000011 0c310101   1 adm1      = 43536531 30202020 43533539 30303032   CSe10 CS590002 adm2      = 5554494d 41434f20 43533539 30303032   UTIMACO CS590002 adm3      = 494e5354 414c4c45 44000000 00000000   INSTALLED </pre>

The displayed fields have the following meaning:

mode	<p>Operating mode of the device</p> <ul style="list-style-type: none"> <li>Operational The regular set of firmware modules ( <code>*.msc</code> ) is started. All administration and cryptographic functions are available.</li> <li>Operational Mode – Administration-Only The regular set of firmware modules ( <code>*.msc</code> ) is started. All administration functions are available. All cryptographic functions are blocked.</li> <li>Maintenance Backup set of firmware modules ( <code>*.sys</code> ) is started (e.g., in alarm state)</li> <li>Bootloader The Bootloader is running. The operating system and the regular set of firmware modules have not been started yet.</li> </ul>
state	<p>Current operating state of the device (should be INITIALIZED):</p> <ul style="list-style-type: none"> <li>MANUFACTURED This state is only relevant during production process.</li> <li>INITIALIZED Firmware modules and all system keys (Production Key, Module Signature Key, default Administrator Key ( <code>ADMIN.key</code> file)) are loaded.</li> <li>DEFECT The device is defect. Contact the manufacturer, Utimaco IS GmbH.</li> </ul>
temp	Current temperature of the device (in °C).
alarm	<p>Current alarm status. It can be either ON or OFF. If ON, the following reasons are possible and shown in case of an alarm state:</p> <ul style="list-style-type: none"> <li>Power is too low (empty battery)</li> <li>Power is too high</li> <li>Temperature too high (&gt; 66°C)</li> <li>Temperature too low (&lt; -13 °C)</li> <li>Outer foil is broken (only relevant for CSe-Series)</li> <li>Inner foil is broken (only relevant for CSe-Series)</li> <li>External Erase is executed (manually by a short-circuit of the corresponding pins on the PCIe card)</li> <li>Invalid/Corrupted Master Key</li> <li>Communication to sensor controller failed</li> </ul> <p>In addition it is shown if the alarm reason is still present or if it has been removed in the meantime (for example, an empty battery has been replaced).</p>

bl_ver	Current bootloader version and the model type of the device
hw_ver	Version of the hardware for the CSe-Series and Se-Series Gen2.
uid	<p>UID is an 8-byte binary data field.</p> <p>The UID is a "Universal Identification" that uniquely identifies every PCIe card. It is stored on a hardware component and loaded onto the PCIe card during production.</p> <p>The UID is displayed when the status information is extracted. It is not stored on the device.</p>
adm1	<p>adm1 is a readable character string, with a length of 16 characters.</p> <p>The first 8 characters of adm1 contain a short form of the device model type, filled with blank spaces CSe10, Se12, CSe100, Se52, Se500 or Se1500.</p> <p>The second 8 characters represent the unique serial number of the PCIe card. This serial number is assigned by Utimaco IS GmbH during manufacture and then loaded into the device. In case of a real hardware PCIe card, the serial number starts with the letters CS, followed by a 6-digit number.</p> <p>The adm1 character string is displayed when you select the status information. The 8-character serial number CSxxxxxx is also stored on the PCIe card.</p> <p>In case of the device Simulator, the serial number has the format SI&lt;xxxxxxx&gt;. SI stands for "simulator" and &lt;xxxxxxx&gt; stands for the port number the simulator is listening on, i.e., 003001 for the first simulator instance, 003003 for the second simulator instance etc. Example: SI003001</p> <p>In case of the PaymentServer Simulator, the serial number is always CS000000.</p>
adm2	<p>adm2 is a readable 16-character string.</p> <p>The contents of the adm2 character string is also assigned by Utimaco IS GmbH and loaded onto the device during production. Whilst the device is being manufactured, the name of the firmware module package loaded for the customer is also recorded here, according to which device model series is being produced.</p> <p>The adm2 character string is displayed when you select the status information. It is not stored on the device.</p> <p>This field may be empty.</p>
adm3	<p>adm3 is a readable 16-character string.</p> <p>During production, a default value is recorded here, according to which device model is being manufactured.</p> <p>For a CSe-Series and Se-Series Gen2, the value INSTALLED is recorded here.</p>
error state	Error code indicating that a power-on self-test has failed. If these tests succeed, nothing is shown.

Table 11: Meaning of the information fields output by the GetState command

The bit representation of the state field has the following meaning:

Bit(s)		Value	Description
	Device state		
0 ... 6		1	DEFECT
		2	MANUFACTURED
		4	INITIALIZED

<b>Bit(s)</b>		<b>Value</b>	<b>Description</b>
		5	OPERATIONAL
	Alarm		
7		0	OFF
		1	ON
	Sensor		
8		0	The temperature is too low.
		1	No temperature low alarm has been registered.
9		0	The temperature is too high.
		1	No temperature high alarm has been registered.
10 This is only possible for CSe-Series.		0	The inner foil has been broken.
		1	No inner foil alarm has been registered.
11 This is only possible for CSe-Series.		0	The outer foil has been broken.
		1	No outer foil alarm has been registered.
12		-	This bit is not used any more.
13		0	The power is too high (power overdrive).
		1	No power high alarm has been registered.
14		0	The power is too low.
		1	No power low alarm has been registered.
15		0	The external erase has been executed.
		1	No external erase alarm has been registered.
16		0	No alarm is present.
		1	An alarm is still present.
17		0	No alarm has occurred.
		1	An alarm has occurred.
	FIPS140 mode		
18		0	FIPS mode OFF
		1	Some FIPS mode (restricted (CryptoServer Se-Series only) or validated, see bit 26 below)
	Boot mode		
19 ... 20		0	The boot loader is started (not possible here).
		1	*.sys modules are started.
		2	*.msc modules are started.
	...		
	FIPS140 (2)		

Bit(s)		Value	Description
26 This bit is only relevant if bit 18 has been set		0	FIPS mode = ON
		1	FIPS Mode = OFF but FIPS restrictions are applied (CryptoServer Se only)
	Administration-only mode		
27		0	Administration-Only Mode = OFF
		1	Administration-Only Mode = ON

#### 4.4.2 SetAdminMode



This command can only be executed with csadm version 1.9.0 or higher, and with CMDS firmware module version 3.3.0.0 or higher.

This command enables the device to switch temporarily between the normal Operational Mode and the restricted Operational Mode – Administration-Only without being restarted. In Operational Mode – Administration-Only only functions needed for the device administration are available, and all cryptographic functions are blocked.

The `SetAdminMode` command can only be executed if the device is currently operating in either Operational Mode or Operational Mode – Administration-Only. It cannot be executed if the device is currently in Maintenance Mode.

The operating mode set with this command is only relevant until the next time the device is restarted. To define the restricted Operational Mode – Administration-Only mode to be active after a restart of the device, perform the `csadm SetStartupMode=1` command.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; SetAdminMode=&lt;mode&gt;</code>
---------------	--

<b>Authentication</b>	Permission 2 in the user group 6 (02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

Parameter	Description
<mode>	<ul style="list-style-type: none"> <li>▪ 1 – Sets the device operating mode to restricted Operational Mode – Administration-Only. All cryptographic services are disabled.</li> <li>▪ 0 – Sets the device operating mode back to Operational. The complete functional interface of the device can be used again.</li> </ul>

<b>Example</b>	Switch the device operating mode temporarily to restricted Operational Mode – Administration-Only: csadm Dev=192.168.1.2 LogonSign=ADMIN, :cs2:cjo:USB0 SetAdminMode=1
----------------	--

<b>Output</b>	Upon successful execution of the command, no return is given.
---------------	---

#### 4.4.3 SetStartupMode



This command can only be executed with csadm version 1.9.0 or higher, and with CMDS firmware module version 3.3.0.0 or higher.

With this command you can disable the automatic activation of the cryptographic interfaces of the device. The device starts, by default, in Operational Mode, which means without any restrictions on the cryptographic functions.

The `SetStartupMode` command defines the operating mode of the device – either Operational Mode or Operational Mode – Administration-Only - after a restart has been executed.

<b>Syntax</b>	csadm [Dev=<device>] SetStartupMode=<mode>
---------------	--

<b>Authentication</b>	Permission 2 in the user group 6 (02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

Parameter	Description
<mode>	<ul style="list-style-type: none"> <li>▪ 1 – After a restart the device starts in Operational Mode – Administration-Only. All cryptographic functions are disabled.</li> <li>▪ 0 – After a restart the device starts in Operational Mode, and all cryptographic functions are available again.</li> </ul>

<b>Example</b>	<code>csadm Dev=PCI:0 LogonSign=adminUsr,:cs2:cjo:USB0 SetStartupMode=1</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

#### 4.4.4 GetStartupMode



This command can only be executed with csadm version 1.9.0 or higher, and with CMDS firmware module version 3.3.0.0 or higher.

This command displays the operating mode – Operational or Operational Mode – Administration-Only - in which the device will boot after the next restart (evtl. previously set with the device command).

- If Operational Mode is displayed this implies that the full functional interface of the device will be available after a restart.
- If Operational Mode – Administration-Only is displayed this implies that the cryptographic interface of the device will be deactivated after a restart, and only administration services will be available. To make the full functional interface of the device temporarily available again, without restarting the device, use the command `SetAdminMode`.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetStartupMode</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=3001@194.168.4.107 GetStartupMode</code>
<b>Output</b>	<p>Upon successful execution of the command, the currently set startup mode of the device is returned.</p> <ul style="list-style-type: none"> <li>▪ Operational Mode (0)</li> <li>▪ Operational Mode – Administration-Only (1)</li> </ul>

#### 4.4.5 GetBattState

This command shows the state of the two device batteries – the Carrier Battery and the External Battery. The state 'low' indicates that the battery should be renewed as soon as possible to avoid a power low alarm.



The External Battery is only relevant for a LAN device. For a PCIe card the External Battery is not present, and thus the state 'absence' for the External Battery is no reason to worry.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetBattState</code>
<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>Example</b>	<pre>csadm Dev=4000@192.168.1.1 GetBattState or csadm Dev=/dev/cs2.0 GetBattState</pre>
<b>Output</b>	<p>Upon successful execution of the command, the current state of the battery is returned.</p> <pre>Carrier Battery: ok (3.055 V) External Battery: absence</pre>



## 4.4.6 StartOS

This command starts the operating system in normal mode, i.e., the regular set of firmware modules ( \*.msc ) will be started. These firmware modules can be managed (loaded, updated or deleted) by the customer.



On startup of the device (power-on, or after the Reset command) the StartOS command is automatically executed by the bootloader if it doesn't receive any command within 3 seconds.

**Syntax**

```
csadm [Dev=<device>] StartOS
```

**Parameter****Description**

&lt;device&gt;

Device address  
This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

**Example**

```
csadm Dev=10.1.7.1 StartOS
```

**Output**

Upon successful execution of the command, no output is returned.

If no operating system SMOS is present (SMOS has not been loaded onto the device before), an error message is displayed.

SMOS checks the integrity of every firmware module. A firmware module is started only after successful verification.

If an error occurs during the startup of a firmware module, the corresponding error message is added to the boot log file.



This command can only be performed in Bootloader Mode. After it is successfully performed, the device is in Operational Mode.

If this command is called in any other but the Bootloader Mode (Operational Mode, Maintenance Mode) it will be ignored.

### 4.4.7 RecoverOS

This command starts the operating system in Maintenance Mode, i.e., the set of backup copies of the firmware modules ( \*.sys ) is started. This set of basic firmware modules has been loaded during the production process and can't be modified or deleted by the customer.



The backup set of the firmware modules has to be started explicitly by the user, using this `RecoverOS` command.

This procedure may be necessary if it is no longer possible to start the regular set of firmware modules ( \*.msc files), for example because one or more indispensable modules (for example, SMOS, CMD5, ADM, UTIL) have been deleted by mistake, or because a defect firmware module has been loaded.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] RecoverOS</code>
---------------	---

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=10.1.7.1 RecoverOS</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is returned.
---------------	--

If no operating system SMOS is present (SMOS has not been loaded onto the device before), an error message is displayed.

SMOS checks the integrity of every firmware module, a firmware module is started only after successful verification.

If an error occurs during the startup of a firmware module, the corresponding error message will be added to the boot log file.



This command can only be performed in Bootloader Mode. After it is successfully performed, the device is in Maintenance Mode.

If this command is called in any other mode but Bootloader Mode (Operational Mode, Maintenance Mode) it will be ignored.

#### 4.4.8 GetTime

This command displays the system time of the device (PCIe card). This time actually has no time zone but it is defined/regarded as a time in the GMT (UTC) time zone. If you use a LAN device and you want to get the time of the LAN host, perform the `csadm CSLGetTime` command instead.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetTime</code>
---------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=PCI:0 GetTime</code>
----------------	--------------------------------------

<b>Output</b>	<p>Upon successful execution of the command, the current time of the device is returned.</p> <pre>date: 06.11.2012 time: 15:35:49.400 &lt;local time&gt; date: 06.11.2012 time: 13:35:49.412 &lt;UTC/internal&gt;</pre> <p>The second line shows the actual time of the device (PCIe card). This time is defined/regarded as a time in the GMT (UTC) time zone. Therefore, it is called (UTC/internal). The first line shows the same time converted into the time zone of the administration computer. Therefore, it is called (local time). This is not necessarily the time of the administration computer.</p> <p>The date is shown in the DD.MM.YYYY format.</p>
---------------	---

The system clock of the device has a resolution of 1/1000 second (one millisecond).

#### 4.4.9 SetTime

This command sets the system clock of the device (PCIe card). This time actually has no time zone but it is defined/regarded as a time in the GMT (UTC) time zone.

If you use a LAN device and you want to set the time of the LAN device host, perform the `csadm CSLSetTime` command instead.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; SetTime=&lt;time&gt;</code>
---------------	---

<b>Authentication</b>	Permission level 2 in user group 6 (02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<time>	<ul style="list-style-type: none"> <li>Set the time manually in format <code>YYYYMMDDhhmmss</code>. YYYY=year, MM=month, DD=day, hh=hour, mm=minute, ss=second. This is a time in the time zone of the administration computer. This time is automatically converted to the GMT (UTC) time zone. This GMT time is assigned to the device time. The time in the time zone of the administration computer mentioned above is not the time of the administration computer. The administration computer is the computer csadm is running on.</li> <li>GMT (recommended) Use the time of the administration computer to set the time of the device. The time of the administration computer has a time zone. This time is automatically converted to the GMT (UTC) time zone. This GMT time is assigned to the device time.</li> </ul>

<b>Example</b>	<pre>csadm LogonSign=ADMIN,:cs2:cjo:USB0 SetTime=GMT csadm LogonPass=sven,ask SetTime=20020602115500</pre>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the current time of the device is returned.</p> <p>The third line shows the actual time of the device (PCIe card). This time is defined/regarded as a time in the GMT (UTC) time zone. Therefore, it is called (UTC/internal).</p> <p>The second line shows the same time converted into the time zone of the administration computer. Therefore, it is called (local time). This is not necessarily the time of the administration computer. The date is shown in the DD:MM.YYYY format.</p> <pre>Time succzssfully set to: date: 06.12.2019   time: 10:05:11.000 (local time) date: 06.12.2019   time: 09:05:11.000 (UTC/internal)</pre>
---------------	---

If the device is in the alarm state, that means, if the `csadm GetState` command delivers among other things the alarm = ON output, the `csadm SetTime` command can only be performed if mutual authentication is not enabled.

Any changes of the clock are taken down on the audit log file. The new entry contains the old time, as well as the new time value. The `csadm GetAuditLog` command can be used in any mode to view this file.



The command is not sent to the device until authentication is done. If this requires a lot of time (for example, insertion of a smartcard and PIN input) there is a gap between the given time and the current time. If the time of the device has to be set very precisely, you can enter a future time and perform the last step (for example, pressing 'OK' after PIN entry on the PIN pad) when this time is reached.

#### 4.4.10 GetBootLog



Log messages can be watched externally by connecting a PC's serial port with a crossed serial cable to the serial port 1 of the device (at the bracket of the PCIe card). Use a terminal tool (e.g. PuTTY) to view the log messages.

On a CSe device, a special USB-To-Serial adapter (Prolific PL2303) has to be connected to one of the USB ports of the device before.

`GetBootLog` returns the boot log file, which contains log messages that are made during the boot process. The log messages are made by the operating system and other firmware modules or by the bootloader if the command is called in Bootloader Mode. The boot log is held in working memory and is not written into a permanent file. In this way the content of the previous boot log file is cleared every time the operating system starts.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetBootLog</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

**Example**

csadm Dev=3001@194.168.4.107 GetbBootLog

**Output**

Upon successful execution, the command returns the boot log.  
The example output may differ from the output on your CryptoServer depending on the installed firmware package and its version.

```
--.---.--- --:---:-- SMOS Ver. 7.1.0.0 (Jan 1 1970) started [0]
--.---.--- --:---:-- Compiler Ver. 9.3.0
--.---.--- --:---:-- CPU clock frequency: 1000000000
--.---.--- --:---:-- trng_init
--.---.--- --:---:-- Sensory Controller Ver. 2.0.0.42 [0/0]
--.---.--- --:---:-- Real Random Number Generator initialized with:
RESEED_INTERVAL = 1000
PREDICTION_RESISTANCE = 0
REALRANDOM_SHARE = 3
--.---.--- --:---:-- Pseudo Random Number Generator initialized
with:
RESEED_INTERVAL = 1000
PREDICTION_RESISTANCE = 0
REALRANDOM_SHARE = 0
--.---.--- --:---:-- Load module 'fips140.msc' from FLASHFILE
--.---.--- --:---:-- FIPS module has no list of FIPS approved
modules!
--.---.--- --:---:-- FIPS strict mode
--.---.--- --:---:-- Signed Licence File found: dev.slf
--.---.--- --:---:-- Load module 'adm.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'cmds.msc' from FLASHFILE
--.---.--- --:---:-- CMDS: .pscf support enabled
--.---.--- --:---:-- Load module 'crypt.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'cxi.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'db.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'hce.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'mbk.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'ntp.msc' from FLASHFILE
--.---.--- --:---:-- Load module 'util.msc' from FLASHFILE
--.---.--- --:---:-- module 0x01 (FIPS140) initialized successfully
--.---.--- --:---:-- module 0x83 (CMDS) initialized successfully
--.---.--- --:---:-- module 0x86 (UTIL) initialized successfully
--.---.--- --:---:-- CSAR XRS Accelerator detected
--.---.--- --:---:-- module 0x0a (HCE) initialized successfully
--.---.--- --:---:-- module 0x9f (CRYPT) initialized successfully
--.---.--- --:---:-- module 0x88 (DB) initialized successfully
--.---.--- --:---:-- MBK: Cannot auto generate key (B0001100)
--.---.--- --:---:-- module 0x96 (MBK) initialized successfully
--.---.--- --:---:-- module 0x68 (CXI) initialized successfully
--.---.--- --:---:-- CMDS: .pscf support enabled
--.---.--- --:---:-- module 0x87 (ADM) initialized successfully
--.---.--- --:---:-- module 0x9a (NTP) initialized successfully
--.---.--- --:---:-- CMDS: all certificates for authentication
successfully loaded.
--.---.--- --:---:-- CMDS: successfully initialized in FIPS mode
```

### 4.4.11 GetAuditLog

The `csadm GetAuditLog` command shows the contents of all audit log files. This command can be performed in any mode and does not need any authentication.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetAuditLog</code>
---------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=3001@194.168.4.107 GetAuditLog</code>
----------------	---

<b>Output</b>	Upon successful execution, the command returns the audit log entries. <pre> __._.__ __:__:__ [] FC:0x000 SFC: SMOS Ver. 7.1.0.0 successfully started __._.__ __:__:__ FC:0x087 SFC:0x07 Set Time [b087000f]</pre>
---------------	--

### 4.4.12 ClearAuditLog

This function erases the content of audit logfiles. For a continuous auditing the newest logfile(s) can be kept.

Consider that there is another `csadm` command for deleting audit log files, `csadm ClearAuditLogFiles`. However, that command can be applied to audit log files with long file names only (for example, audit00003A.log).

You can verify which audit log files are present in the FLASH memory of the CryptoServer by performing a command according to the following example:

```
csadm Dev=3001@127.0.0.1 ListFiles | grep .log
```

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; ClearAuditLog[=&lt;n&gt;]</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 6 or 7 (e.g. 02000000).
-----------------------	--

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<n>	n newest logfiles will not be deleted

<b>Example</b>	csadm LogonSign=ADMIN,:cs2:cjo:USB0 ClearAuditLog=2
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

#### 4.4.13 GetAuditConfig

This command shows the configuration setting for the audit functionality of the device.

<b>Syntax</b>	csadm [Dev=<device>] GetAuditConfig
---------------	-------------------------------------

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	csadm GetAuditConfig
----------------	----------------------

<b>Output</b>	Upon successful execution of the command, the audit log configuration parameters are returned. Number of logfiles: 3 Rotate logfiles: yes Max filesize: 200000 Events: 0x0000073F (Bits 1:2:3:4:5:6:9:10:11)
---------------	--

<b>Output</b>	Upon successful execution of the command, the audit log configuration parameters are returned. Number of logfiles: 3 Max filesize: 200000 Events: 0x0000073F (Bits 1:2:3:4:5:6:9:10:11)
---------------	--

The audit log configuration parameters have the following meaning.



<b>Number of log files</b>	<b>Maximal number <math>n</math> of audit log files (<math>2 \leq n \leq 10</math> allowed)</b>
Rotate logfiles	<ul style="list-style-type: none"> <li>yes (default) Audit log files will be written rotatingly, i.e., once all existing audit log files are full, the device starts overwriting the first (oldest) one, etc.</li> <li>no Audit log files will NOT be written rotatingly, i.e., once all existing audit log file are full, the device refuses executing any auditable functions except for deleting the audit log file.</li> </ul>
Max. filesize	Maximum length of one audit log file in bytes
Events	Audit message class mask of the configured audit message classes: An event with audit message class number $n$ will be audited if and only if the bit $n$ is set. In the example above (audit message class mask 0x00000007, i.e., bits 1, 2 and 3 set), all events from the audit message classes firmware/file management, user management and time management will be audited.

Table 12: Audit configuration parameters

<b>Number of log files</b>	<b>Maximal number <math>n</math> of audit log files (<math>2 \leq n \leq 10</math> allowed)</b>
Max. filesize	Maximum length of one audit log file in bytes
Events	Audit message class mask of the configured audit message classes: An event with audit message class number $n$ will be audited if and only if the bit $n$ is set. In the example above (audit message class mask 0x00000007, i.e., bits 1, 2 and 3 set), all events from the audit message classes firmware/file management, user management and time management will be audited.

Table 13: Audit configuration parameters

#### 4.4.14 SetAuditConfig

For an overview of all auditable events, see [Configuring Auditable Events \(p. 21\)](#). With this command the configuration of the audit functionality of the device can be changed.

<b>Syntax</b>	csadm [Dev=<device>] <Authentication> ... SetAuditConfig=<param1=value1>[,<param2=value2>,<param3=value3>, ...]
---------------	---

<b>Authentication</b>	Permission level 2 in user group 6 and 7 (22000000)
-----------------------	---

Parameter	Description
<device>	<p>Device address</p> <p>This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.</p>
<param>	<p>Configuration parameter</p> <ul style="list-style-type: none"> <li>▪ <b>MaxFileCount</b> Maximal number <math>n</math> of audit log files that may exist at a point in time on the device (<math>2 \leq n \leq 10</math> allowed)</li> <li>▪ <b>RotateLogfiles</b> <ul style="list-style-type: none"> <li>• yes (set by default; strongly recommended) Audit log files will be written rotatingly. If the device is in the Maintenance Mode and the <code>*.sys</code> files originate from a device version <math>&lt; V4.30</math>, 'rotatingly' means that once all existing audit log files are full, the device starts overwriting the first (oldest) one, etc. However, if the device is in the Maintenance Mode and the <code>*.sys</code> files originate from a device version <math>\geq V4.30</math> or if the device is in the Operational Mode, 'rotatingly' means that once all existing audit log files are full, the device deletes the first (oldest) one and creates a new audit log file with the next long audit log file name.</li> <li>• no Audit log files will NOT be written rotatingly, i.e., once all existing audit log files are full, the device refuses executing any functions that are able to create an audit log entry. The only exceptions are functions for deleting audit log files. If an audit log file is deleted, this is logged as the first new audit log entry.</li> </ul> </li> <li>▪ <b>MaxFileSize</b> Maximum size of one of the audit log files in byte (only <math>4,000 \text{ byte} \leq \text{len} \leq 240,000 \text{ byte}</math> allowed)</li> <li>▪ <b>Events</b> Class mask of the audit message classes to be configured. Coded either as a list of bits (the audit message class numbers) separated by a colon (for example, '1:2:3'), or as an eight digits long hexadecimal (same example: 0x00000007): An event with audit message class number <math>n</math> will be audited, if and only if the bit <math>n</math> is set. If e.g. audit message class mask 0x00000007 is set, i.e. bits 1, 2 and 3 are set, all events from the audit message classes firmware/file management, user management and time management will be audited.</li> </ul>
<b>Example</b>	<pre>csadm LogonPass=svenpaul,swordfish ... SetAuditConfig=MaxFileCount=5,RotateLogfiles=yes,Events=1:2:3 csadm LogonSign=nils,:cs2:cjo:USB0 ... SetAuditConfig=MaxFileCount=3,MaxFileSize=200000,Events=0x00000007</pre>

**Output**

Upon successful execution of the command, no output is given.



If a configuration parameter is not set, it remains unchanged.

## 4.4.15 GenerateAuditLogKey

This command generates an audit log signature key in the `auditkey.db` database and outputs its public part on the console. The complete output can be redirected into a file, which can be used as input for the `csadm GetSignedAuditLog` and `csadm VerifySignedAuditLog` commands.

If `auditkey.db` already exists, an error is returned. Use the `csadm DeleteFile=` command to delete the database/key. The database is also deleted if an alarm has occurred, an External Erase or a `csadm ... Clear=INIT` command has been performed.

The `auditkey.db` database can only contain one audit log signature key. If you want to use a different audit log signature key, use the `csadm DeleteFile=` command to delete the database/key and perform the `csadm GenerateAuditLogKey` command to generate a new key in a new `auditkey.db` database.

Consider that if you delete the `auditkey.db` database, you cannot sign any audit log files with the key stored in this database anymore and that any audit log files previously signed with this key cannot be verified unless you have backed up the `auditkey.db` database.

The `auditkey.db` database can be backed up and restored using the `csadm BackupDatabase` and `csadm RestoreDatabase` commands.

If the device is part of a cluster, ensure that the generated audit log signature key is distributed in the entire cluster by using the `csadm BackupDatabase` command and the `csadm DatabaseRestore` command.

The `csadm GenerateAuditLogKey` command is supported as of the device version 4.30 and for ADM version  $\geq 3.0.26.0$  and  $4.6.1.0 \leq \text{SMOS version} < 5.0$  or  $\text{SMOS version} \geq 5.6.1.0$ .

The `csadm GenerateAuditLogKey` command is supported in Operational Mode only. Perform the `csadm GetState` command to retrieve the mode of the device.

**Syntax**

```
csadm [Dev=<device>] <Authentication> GenerateAuditLogKey=<mode>
```

<b>Authentication</b>	Permission level 2 in user group 6 and 7 (22000000)
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<mode>	Mode of the audit log signature key <ul style="list-style-type: none"> <li>1 3072-bit RSA key with the public exponent 0x10001, PKCS#1 RSA PSS signature with SHA-256 and a 32-byte salt length</li> <li>2 ECDSA with NIST P-256 curve and SHA-256</li> </ul>

<b>Example</b>	<p>Example 1: Generate an RSA audit log signature key on the (LAN) device at address 192.168.4.1 and print the output on the console.  csadm Dev=192.168.4.1 LogonSign=ADMIN,:cs2:cjo:USB0  GenerateAuditLogKey=1</p> <p>Example 2: Generate an RSA audit log signature key on the (LAN) device at address 192.168.4.1 and store the public key into the ./pub_audit_log_key.txt file.  csadm Dev=192.168.4.1 LogonSign=ADMIN,:cs2:cjo:USB0  GenerateAuditLogKey=1 &gt; ./pub_audit_log_key.txt</p>
----------------	---

**Output**

Upon successful execution of the command, the Audit Log Key is returned.  
 The output has the format <device serial number>=<public audit log signature key>  
 CS000000=01015554494D41434F204353303030303030C29F4C04C1CD284E01809  
 673B455A86EB019FA9FD37AEFDE119FC3A3C4E7A90D9AD25DED11926F4B0B7A096  
 CEF7A3949D6177071DC49F7C6BF86B3B72C987AAD75D6CA69ACAB5C657D3A4BE4E  
 9BBE41BC3D4D02D7A9B515E2D5B0BC2FE3EA79E6A1BD39C91E37259D32C93C36B8  
 2CD69FB4D7FD1544D7C1DD6992C1E92D7B9B626AAC755901E1B95BBD096AAB15DB  
 C28BFB58ABFFF051DF468741AB8723E711E4D6AED05B2C57CF0C3E37AECAF4727C  
 66C0C81768C24399836F90FF22FE24BEB0A76A01C12EFE9E833E219F0114484E99  
 453491CAC59BD95BB25F912CC3332592C4FD6D9FB9412B5A0E76010C20E7D12495  
 D860AB58B2C13A92CB8BC876065B89256CF5379A6269377DCB13774CF3B17CCAA8  
 F4611161A1CF2555300C30912F585D89D6455C517216E7A23A23BFF3209CD887E6  
 4A6DF5F200C7BA020CE5506A47FCF248572D883569E7504C4C78F462358379D8A5  
 7CD5478D1BEB9960F72AE9E1A53EEEDCFC3EBC217E9EC3A47D97518E08C9202E51  
 3242E255081AAE22C805DFAF2B1578FC7BE46A52101807485F424DACF65C743E07  
 EEDDB90D9DC827D2FF2DEC14C96713B6F8EE0F48853BFFDE19CD375B438DDE15F3  
 D6BC3BAB55514A2FB2DA8FE3E1AE2EE5CEADDCBE50D91B5C585DE6BB18584EB484  
 1F70D35C139BE971ACDFAEE6AD11601571B157BD311C2365D9F970D166CD3456E2  
 4A8E19EAA4EC8631C01D1CC796E51E9194E395EDD74FE0466133F5863302AAFF1E  
 ED8B7EF571822C8DC1E469E944C34B7BB853D20A171AD20D53BD70D98F5EBAC63C  
 A1D7FB37D8C2C703BB2D3255D79D2B81A4D1F887736D5B0F3589E181FE2DDB3FB6  
 15D1942A33637F62C03AA6F1A450DC3ADE25C5C76A0506F009A603673DAF3D913  
 DF0398951D10AF493E82DE325CACB49A5B00BAEBC9E68354BC59E183129DD5BBC0  
 2F072AA8592C0BFF98FE3645F8719D6879E7412F278CB938B0A525D828B5C74636  
 AC7D2E7D399CEDE409EC3BC0C25B0D255859EB961B6AECFE87842B29E57503B6D5  
 210F83FD4F0234FECE9CBF70B46C7EC556A5647790E6677684CB297876B0024752  
 BF474FD0CD7676D569514

You can verify whether the `auditkey.db` has been created by performing a command according to the following example:

```
csadm Dev=3001@127.0.0.1 ListFiles | grep auditkey.db
```

Example output:

```
FLASH\auditkey.db          1785      -
```

If you want to change the audit log signature key, you must first delete the existing audit log signature key by performing, for example, the following command:

```
csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,ADMIN.key DeleteFile=auditkey.db
```

Then generate a new audit log signature key by performing the `csadm GenerateAuditLogKey` command.

#### 4.4.16 GetAuditLogKey

This command gets the audit log signature key from the `auditkey.db` database and outputs its public part. This audit log signature key must have been generated before using the `csadm GenerateAuditLogKey` command.

The `csadm GetAuditLogKey` command does not create any audit log entry.

The `csadm GetAuditLogKey` command is supported as of the device version 4.30 and for ADM version  $\geq 3.0.26.0$  and  $4.6.1.0 \leq \text{SMOS version} < 5.0$  or SMOS version  $\geq 5.6.1.0$ .

The `csadm GetAuditLogKey` command is supported in Operational Mode only. Perform the `csadm GetState` command to retrieve the mode of the device.

The last part of the `csadm GetAuditLogKey` command output is the signature of the first part. As a consequence, the last part changes for each execution of this command. Only the first part is identical to the first output part of previous calls of this command or of the `csadm GenerateAuditLogKey` command.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GeAuditLogKey</code>
---------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<p>Example 1: Get the current audit log signature key from a (LAN) device at the IP address 192.168.4.1 and print the output on the console. <code>csadm Dev=192.168.4.1 GetAuditLogKey</code></p> <p>Example 2: Get the current audit log signature key from a (LAN) device at the IP address 192.168.4.1 and redirect the output into the <code>./pub_audit_log_key.txt</code> file. <code>csadm Dev=192.168.4.1 GetAuditLogKey &gt; ./pub_audit_log_key.txt</code></p>
----------------	---

<b>Output</b>	<p>Upon successful execution of the command, the Audit Log Key is returned. The output has the format &lt;device serial number&gt;=&lt;public audit log signature key&gt; CS000000=01015554494D41434F2043533030303030C29F4C04C1CD284E01809673B455A86EB019FA9FD37AEFDE119FC3A3C4E7A90D9AD25DED11926F4B0B7A</p>
---------------	---

#### 4.4.17 GetSignedAuditLog

This command retrieves all signed audit log files from the device.

The `csadm GetSignedAuditLog` command does not create any audit log entry.

The `csadm GetSignedAuditLog` command is supported as of the device version 4.30 and for ADM version  $\geq 3.0.26.0$  and  $4.6.1.0 \leq \text{SMOS version} < 5.0$  or SMOS version  $\geq 5.6.1.0$ .

The `csadm GetSignedAuditLog` command is supported in Operational Mode only. Perform the `csadm GetState` command to retrieve the mode of the device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; [AuditPubKey=&lt;keyfile&gt;] GetSignedAuditLog=&lt;outdir&gt;</code>
---------------	---

<b>Authentication</b>	Permission level 1 in user group 6 (01000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<keyfile>	Path to the file containing the public part of the audit log signature key. The output of the <code>csadm GenerateAuditLogKey</code> command or the <code>csadm GetAuditLogKey</code> command must have been redirected into this keyfile before. If the CS_AUDIT_KEYS environment variable has been set to this path, the keyfile parameter can be omitted. If this file does not exist, an error message is shown.
<outdir>	Path to the output directory on the computer csadm is running on (host) to store the signed audit log files. The content of the log files is not encrypted but signed. Each signed audit log file is named according to the schema <serial number> <file number>.log and stored in the specified output directory. <serial_number> is the serial number of the device and <file_number> is an 8-digit hexadecimal number without a leading 0x. Example name of an audit file on the host: CS123456_0013AC97.log  Consider that the audit log files on the device are usually named audit<file number>.log and stored in the FLASH directory. In this case, <file_number> is a 6-digit hexadecimal number without a leading 0x. Perform the <code>csadm ListFiles</code> command to retrieve the names of the audit log files on the device. Example name of an audit log file on the device. audit13AC97.log

<b>Example</b>	Retrieve the audit log files from the device (LAN) at address <code>192.168.4.1</code> and store them in the <code>./logs</code> directory. To do so, use the public part of the audit log signature key stored in the <code>./pub_audit_log_key.txt</code> file. <code>csadm Dev=192.168.4.1 LogonSign=ADMIN,ADMIN.key AuditPubKey=./pub_audit_log_key.txt GetSignedAuditLog=./logs</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is returned.
---------------	--

#### 4.4.18 VerifySignedAuditLog

This command verifies a specified signed audit log file. To perform this command, no connection to the device is required.

The `csadm VerifySignedAuditLog` command does not create any audit log entry.

The `csadm VerifySignedAuditLog` command is supported as of the device version 4.30 and for ADM version  $\geq 3.0.26.0$  and  $4.6.1.0 \leq \text{SMOS version} < 5.0$  or SMOS version  $\geq 5.6.1.0$ .

The `csadm VerifySignedAuditLog` command is supported in Operational Mode only. Perform the `csadm GetState` command to retrieve the mode of the device.

<b>Syntax</b>	<code>csadm [AuditPubKey=&lt;keyfile&gt;] VerifySignedAuditLog=&lt;file&gt;[,print]</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<keyfile>	Path to the file containing the public part of the audit log signature key. The output of the <code>csadm GenerateAuditLogKey</code> command or the <code>csadm GetAuditLogKey</code> command must have been redirected into this keyfile before. If the CS_AUDIT_KEYS environment variable has been set to this path, the keyfile parameter can be omitted. If this file does not exist, an error message is shown.
<file>	Path to the signed audit log files to be verified. This file must have been retrieved before using the <code>csadm GetSignedAuditLog</code> command.
<print>	If set, output the audit log content in the same format as in the <code>csadm GetAuditLog</code> command.

<b>Example</b>	Verify an audit log file of the (LAN) device at address 192.168.4.1. To do so, use the public part of the audit log signature key stored in the <code>./pub_audit_log_key.txt</code> file. <code>csadm Dev=192.168.4.1 AuditPubKey=./pub_audit_log_key.txt VerifySignedAuditLog=./logs/CS123456_0013AC97.log,print</code>
----------------	--



**Output**

```

Upon successful execution of the command, the verification process is returned.
27.09.16 09:34:47 SMOS Ver. 5.4.6.0 successfully started
27.09.16 10:14:17 'ADMIN' authentication(0) failed,
failure counter: 1 [b0830013]
27.09.16 10:14:56 'ADMIN' authentication(0) failed,
failure counter: 2 [b0830013]
27.09.16 10:15:21 'ADMIN' authentication(0) failed,
failure counter: 3 [b0830013]
27.09.16 10:19:10 'ADMIN' authentication(0) failed,
failure counter: 4 [b0830013]
27.09.16 10:23:37 [ADMIN] FC:0x083 SFC:0x06 Change User
'ADMIN' [0]
27.09.16 10:25:41 [ADMIN] FC:0x096 SFC:0x04
mbk key generate: AES-32 'TSMBK' 2-out-of-3 [0]
27.09.16 10:25:41 [ADMIN] FC:0x096 SFC:0x05 mbk_key_import:
AES(32) 'TSMBK', 2 parts, slot 3 [0]
27.09.16 10:32:52 [ADMIN] FC:0x083 SFC:0x0E Add User 'TS-
S01' (0,00000100) [0]
27.09.16 10:33:54 [ADMIN] FC:0x083 SFC:0x0E Add User 'TS-
S02' (0,00000100) [0]
27.09.16 10:36:29 [ADMIN] FC:0x087 SFC:0x07 Set Time from
160927083609Z [0]
27.09.16 12:35:56 [TS-S01] FC:0x083 SFC:0x06 Change User
'TS-S01' [0]
27.09.16 12:36:18 [TS-S02] FC:0x083 SFC:0x06 Change User
'TS-S02' [0]
Error message if the verification fails because an incorrect audit log signature key has
been used for the verification:
Error B9063011
CryptoServer admin library
Admin Module Command Interface
The key was not found

```

#### 4.4.19 ClearAuditLogFiles

This command unconditionally deletes specified audit log files (with long audit log file names only) on the device.

It does not delete signed audit log files with extra long file names that have been retrieved and stored on the computer (host) csadm is running on by performing the `csadm GetSignedAuditLog` command.

The `csadm ClearAuditLogFiles` command is supported as of the device version 4.30 and for ADM version  $\geq 3.0.26.0$  and  $4.6.1.0 \leq \text{SMOS version} < 5.0$  or SMOS version  $\geq 5.6.1.0$ .

The `csadm ClearAuditLogFiles` command is supported in Operational Mode only. Perform the `csadm GetState` command to retrieve the mode of the device.

Consider that there is another csadm command for deleting audit log files, `csadm ClearAuditLog`. That command can be applied to audit log files with short file names (for example, `audit_00.log`) and long files names (for example, `audit00003A.log`).

You can verify which audit log files are present in the FLASH memory of the device by performing a command according to the following example:

```
csadm Dev=3001@127.0.0.1 ListFiles | grep .log
```

Example output:

```
FLASH\audit000000.log      747      -
```

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; ClearAuditLogFiles=&lt;n&gt;[,&lt;outdir&gt;]</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 6 or 7 (i.e. 02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<n>	<ul style="list-style-type: none"> <li>all All the audit log files are deleted.</li> <li>n Hexadecimal number without the leading 0x or leading zeros. Number of an audit log file, for example 3a in the <code>audit00003a.log</code> file name. All audit log files up to and including the audit log file with the specified number are deleted. If the <code>outdir</code> parameter is specified and if the nth file in the <code>outdir</code> directory is identical to the nth file on the device, the nth file on the device is deleted as well. If they differ, the nth file remains untouched.</li> </ul>

Parameter	Description
<outdir>	<p>The path to the directory where the signed audit log files have been stored using the <code>csadm GetSignedAuditLog</code> command. This directory is on the computer (host) <code>csadm</code> is running on. The extra long names of the audit log files in this directory have been built according to the <code>&lt;CryptoServer serial_number&gt; &lt;8-digit hexadecimal number&gt;.log</code> schema. Example file name: <code>CS123456_0013AC97.log</code></p> <p>If the <code>&lt;outdir&gt;</code> directory is specified, it is verified whether the contents of the audit log files in the FLASH memory on the device specified by <code>&lt;n&gt;</code> is identical to the corresponding audit log files in the <code>&lt;outdir&gt;</code> directory on the host. If this is the case, the specified audit log files on the device are deleted. If one or more files are missing or do not match, no file is deleted.</p> <p>The names of the audit log files on the device differ in any case from the names of the corresponding files on the host (long file names vs. extra long file names). The all value cannot be combined with the <code>&lt;outdir&gt;</code> parameter.</p>

Example	<p>Delete all audit log files up to and including <code>audit0000003a.log</code> in the <code>./logs</code> directory from the (LAN) device at address <code>192.168.4.1</code>.</p> <pre>csadm Dev=192.168.4.1 LogonSign=ADMIN,ADMIN.key ClearAuditLogFiles=3a,./logs</pre>
---------	--

Output	<p>Upon successful execution of the command, no output is given.</p> <p>If the <code>&lt;outdir&gt;</code> directory has been specified and if one or more files are missing or do not match (see above), the following error message appears:</p> <pre>Error B0870092 CryptoServer module ADM Signed Audit Log section ClearAuditLogFiles did not delete files, because the hash does not fit to the latest log files hash</pre>
--------	---

#### 4.4.20 MemInfo

This function displays information about the current usage of the non-volatile memory of the CryptoServer.

The desired directory (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	<code>csadm [Dev=&lt;device&gt;] MemInfo[=&lt;dir&gt;]</code>
--------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<dir>	Directory on the device (FLASH or NVRAM)

<b>Example</b>	csadm MemInfo=FLASH
----------------	---------------------

<b>Output</b>	Upon successful execution of the command, information about the current memory usage is returned. max_size = 33292288 used_size = 1832960 free_size = 30753280 available_size = 31459328
---------------	--

The displayed information has the following meaning:

<b>Parameter</b>	<b>Description</b>
max_size	Maximum size of the directory
used_size	Currently used size
free_size	Current free size regarding only already formatted blocks. This value is returned only for diagnostic purposes. It does only show a part of the space that is 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

Table 14: Meaning of the output parameters of csadm MemInfo

#### 4.4.21 Test

With this command, a communication test (echo loopback test) with the device is executed. For each loop it sends random test patterns to the device, beginning with the minimum data length, which will be incremented until the maximum data length is reached. The device echoes the received command. On the host side, the received answer is compared with the command originally sent.

<b>Syntax</b>	csadm [Dev=<device>] Test=[<datalength>,<loopcount> csadm [Dev=<device>] Test=<min_datalength>,<max_datalength>[,<increment>],<loopcount>
---------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<datalength>	Data length [bytes] of the used pattern. <ul style="list-style-type: none"> <li>▪ if a value is set: <code>min_datalength = max_datalength = value</code></li> <li>▪ f omitted: <code>min_datalength = 0,</code> <code>max_datalength = 131072 (128kB)</code></li> </ul>
<min_datalength>	Minimum data length in bytes of the used pattern
<max_datalength>	Maximum data length in bytes of the used pattern (max: 131072)
<increment>	Data length increment [bytes]; default = 1
<loopcount>	Number of execution cycles

<b>Example</b>	<ul style="list-style-type: none"> <li>▪ <code>csadm Test=256,10000</code></li> <li>▪ <code>csadm Test=0,128,2,10</code></li> </ul>
----------------	---



A little time is needed to create the test patterns for each loop. A pure benchmark test leads to slightly better results.

<b>Output</b>	<p>Upon successful execution of the command, the result data of the executed test command is given.</p> <pre> Data type : random data length (min): 256 data length (max): 256 increment : 1 loop count : 1000 len count 256 1000 execution time : 480 ms data throughput : 533333 bytes/s transaction time : 0.480000 ms </pre>
---------------	--

## 4.4.22 ResetAlarm

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

If an alarm occurs on the device, the Master Key and other sensitive data will be erased and an entry will be made into the audit log file. The occurrence of the alarm will be stored as a special alarm state. Even if the physical reason for an alarm has been removed (for example, a low battery was exchanged), the alarm state is still active and has to be manually reset by an authenticated user with `ResetAlarm`. With the command `ResetAlarm` also a new Master Key will be generated.

In alarm state the device is running in Maintenance Mode (i.e., only the set of backup copies of the base firmware modules ( `*.sys` ) is running) and there is no secret or private key left. Since the firmware module DB for database management is blocked, no further keys can be loaded in alarm state. Most device functionality is blocked.

On reset of a pending alarm a new Master Key will be generated and the device will be restarted. Afterwards the device is running in Operational Mode again.



If the reason for the alarm is still pending (for example, temperature still too high), any attempt to reset the alarm state will cause the automatic execution of a reset of the device (in the same way the Reset command does).

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] [&lt;Authentication&gt;] ResetAlarm</code>
---------------	---

<b>Authentication</b>	Permission level 2 in user group 6 or 7 (i.e. 02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 ResetAlarm</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

If the reason for the (physical) alarm is still present, an error message is returned and the device does not leave the alarm state. The device stays in the Maintenance Mode and any new alarm is written into the `audit*.log` file, together with the date and time when the alarm appeared. The `csadm GetAuditLog` command can be used to view this file.

Under certain circumstances audit log file names `audit_<2-digit hexadecimal number>.log` instead of `audit<6-digit hexadecimal number>.log` are used in the Maintenance Mode. If this is the case and if the `csadm ResetAlarm` command has been performed successfully, the CryptoServer enters the Operational Mode after the restart and all `audit_<2-digit hexadecimal number>.log` files are renamed to `audit<6-digit hexadecimal number>.log` files.

### 4.4.23 GetModel

This command displays the model information (hardware and software architecture) of the device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetModel</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=192.168.4.101 GetModel</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, the model of the device is returned. <code>CryptoServer Se-Series Gen2</code> i.e., CryptoServer Se-Series with bootloader version 5.0.0.1 or later <code>CryptoServer CSe-Series</code> i.e., CryptoServer CSe-Series with bootloader version 4.0.1.2 or later
---------------	---

### 4.4.24 Reset

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

After executing Reset it takes up to 15 seconds until the bootloader window is timed out and the operating system is restarted again.

Therefore, we strongly recommend NOT using the Reset command but to use instead

- `Restart` to restart the device as fast as possible (3 – 5 sec) or
- `ResetToBL` to reset the device and to get into Bootloader Mode.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] Reset[=&lt;password&gt;]</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <password>
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<password>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . You can find detailed information about the content of the <code>csxlan.conf</code> file in <i>The Configuration File csxlan.conf</i> in the <i>CryptoServer LAN V5 - Administration Manual</i> . <ul style="list-style-type: none"> <li>▪ for hidden password entry the string 'ask' is used We strongly recommend using a hidden password entry.</li> <li>▪ root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm Dev=192.168.4.101 Reset=password</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, a hardware reset is performed.
---------------	--

#### 4.4.25 ResetToBL

This command will reset the device and get it into Bootloader Mode .

First a reset of the device is performed in the same way as using the `Reset` command. Immediately after the `Reset` command a dummy command is sent to the device. The bootloader now remains active and will not start the operating system SMOS. Thus, the device is in Bootloader Mode now.





If the device is in Bootloader Mode, the operating system (SMOS) and all other firmware modules can be started using the `StartOS` command. This way the device can be put into Operational Mode again.

The `RecoverOS` command on the other hand starts the recovery copies of the firmware modules, which were loaded into the device during production.

In Bootloader Mode no application (no other firmware modules) is running.

**Syntax**

```
csadm [Dev=<device>] ResetToBL[=<password>]
```

**Authentication**

Root password of the LAN device, see parameter `<password>`

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . You can find detailed information about the content of the <code>csxlan.conf</code> file in <i>The Configuration File csxlan.conf</i> in the <i>CryptoServer LAN V5 - Administration Manual</i> . <ul style="list-style-type: none"> <li>for hidden password entry the string 'ask' is used We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

**Example**

```
csadm Dev=192.168.4.101 ResetToBL=password
```

**Output**

Upon successful execution of the command, a reset to Bootloader Mode is performed.

In Bootloader Mode any command sent to the device is responded by the bootloader.

The bootloader remains active until the device is reset again or the `StartOS` or `RecoverOS` command is sent to the device.

This command will reset the device and get it into Bootloader Mode .

First a reset of the device is performed in the same way as using the `Reset` command. Immediately after the `Reset` command a dummy command is sent to the device. The

bootloader now remains active and will not start the operating system SMOS. Thus, the device is in Bootloader Mode now.



If the device is in Bootloader Mode, the operating system (SMOS) and all other firmware modules can be started using the `StartOS` command. This way the device can be put into Operational Mode again.

The `RecoverOS` command on the other hand starts the recovery copies of the firmware modules, which were loaded into the device during production.

In Bootloader Mode no application (no other firmware modules) is running.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] ResetToBL[=&lt;password&gt;]</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <code>&lt;password&gt;</code>
-----------------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . <ul style="list-style-type: none"> <li>for hidden password entry the string 'ask' is used We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm Dev=192.168.4.101 ResetToBL=password</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, a reset to Bootloader Mode is performed.
---------------	--

In Bootloader Mode any command sent to the device is responded by the bootloader.

The bootloader remains active until the device is reset again or the `StartOS` or `RecoverOS` command is sent to the device.

## 4.4.26 Restart

This command will reset/restart the device and get it into Operational Mode.

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



Executing the `Restart` command is the fastest way to restart the operating system of the device. If it has been successfully executed, the device is in Operational Mode afterwards.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] Restart[=&lt;password&gt;]</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <code>&lt;password&gt;</code>
-----------------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . You can find detailed information about the content of the <code>csxlan.conf</code> file in <i>The Configuration File csxlan.conf</i> in the <i>CryptoServer LAN V5 - Administration Manual</i> . <ul style="list-style-type: none"> <li>for hidden password entry the string 'ask' is used We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm Dev=192.168.4.101 Restart=password</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, a restart is performed.
---------------	---

As a general rule, the device has to be restarted after the loading (or replacement) of a new firmware module: The firmware module only becomes active after the device has been restarted.

#### 4.4.27 GetInfo



See the [CryptoServer - Troubleshooting \(p. 243\)](#) document for details about further diagnostic information to be provided to the support team.

This command retrieves information from the PCI/PCIe driver of the device and is used for diagnostic purposes in error case. It will be executed even if the device does not respond to any other command.

The information provided by the `GetInfo` command might help to identify low level problems of the device. However, the interpretation of the output requires extensive knowledge of the device and is not explained in detail here.



Please prepare the output of the `csadm GetInfo` command in case of a support request.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetInfo</code>
<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>Example</b>	<code>csadm Dev=192.168.4.101 Restart=password</code>

<b>Output</b>	<p>Upon successful execution of the command, several information parameters are returned.</p> <p>Example output for a Se-Series Gen2 in a Windows host system:</p> <pre> vers  2.4.1.0 (15.07.2015) fwver 5.1.0.6 slot  0 model 5 state idle spm   00000000 irq   b    1    f    f txlen 00000010 00000000 rxlen 00000064 00000000 count 00000001 00000001 crc   00000000 00000000 </pre> <p>Example output for a CSe-Series in a Linux host system:</p> <pre> vers  3.2.13 slot  0000:02:00.0 model 4 state idle spm   00000000 count      1      1 crc        0      0 irq   b    1    f    f txlen 00000010 00000000 rxlen 00000064 00000000 fwver 1.1.0.7... </pre>
---------------	---

The following table explains selected information fields that are part of the command output. The output fields missing in this table provide device internal information including, among other things, details about the device driver, the slot number etc. that can only be interpreted by the manufacturer Utimaco IS GmbH. Have this information at hand, if you need to contact our support team.

<b>vers</b>	Version of PCI/PCIe driver installed on the Linux or Windows host system
<b>fwver</b>	Version of the device control logic unit FPGA (Field Programmable Gate Array)
<b>slot</b>	PCIe slot number of the host computer where the PCIe card is mounted
<b>model</b>	Device series: ■ 4: CryptoServer CSe-Series ■ 5: CryptoServer Se-Series Gen2
<b>state</b>	Status of the device driver (host side): ■ idle: Ready ■ wait/read/write: Expecting a response/acknowledgment from the device

Table 15: : Meaning of selected information fields of the GetInfo command output

#### 4.4.28 MemInfoCSV

This command displays memory information about all directories (FLASH and NVRAM) in CSV format. The command allows to monitor the memory usage over a longer period of time, e.g. to recognize if there are memory leaks.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] MemInfoCSV</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=192.168.1.2 MemInfoCSV</code>
----------------	---

<b>Output</b>	<code>&lt;time&gt;,&lt;dir1&gt;,&lt;dir1_max_size&gt;,&lt;dir1_used_size&gt;,&lt;dir1_free_size&gt;,&lt;dir1_available_size&gt;,&lt;dir2&gt;,&lt;dir2_max_size&gt;,&lt;dir2_used_size&gt;,&lt;dir2_free_size&gt;,&lt;dir2_available_size&gt;,...</code>
---------------	---

See also `csadm MemInfo`

#### 4.4.29 RamInfo

`RamInfo` displays information about the current usage of the memory types SD-RAM and Secure RAM.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] RamInfo</code>
---------------	---

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER

<b>Example</b>	Display memory information of the device at address 192.168.1.2 csadm Dev=192.168.1.2 RamInfo  Display memory information of the device whose address is defined in the environment variable CRYPTOSERVER csadm RamInfo
----------------	---

<b>Output</b>	CryptoServer Internal Memory Information:				
	type	used_blocks	used_bytes	free_blocks	free_bytes
	SD-RAM Secure	107 80	1079920 28272	11 9	65766752 1003872

### 4.4.30 RamInfoCSV

RamInfoCSV displays information about the current usage of the memory type SD-RAM and Secure in CSV format. The command allows to monitor the memory usage over a longer period of time, e.g. to recognize if there are memory leaks.

<b>Syntax</b>	csadm [Dev=<device>] RamInfoCSV
---------------	---------------------------------

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER

<b>Example</b>	Display memory information of the device at address 192.168.1.2 in CSV format csadm Dev=192.168.1.2 RamInfoCSV
----------------	---

<b>Output</b>	<time>,<sd_used_blocks>,<sd_used_bytes>,<sd_free_blocks>,<sd_free_bytes>,<secure_used_blocks>,<secure_used_bytes>,<secure_free_blocks>,<secure_free_bytes>
---------------	--

See also [csadm RamInfo](#)

### 4.4.31 GenRandom

This command generates random data of given size.

<b>Syntax</b>	<code>csadm GenRandom=&lt;size&gt;[,&lt;filename&gt;]</code>
<b>Parameter</b>	<b>Description</b>
<code>&lt;size&gt;</code>	byte size of random data to be generated
<code>&lt;filename&gt;</code>	output filename where the generated random data will be written to; if omitted, the generated random data will be written to the standard output
<b>Example</b>	<pre>csadm GenRandom=64 csadm GenRandom=200,c:\test\random.txt</pre>
<b>Output</b>	<pre>c:\Utimaco\CS_4.31\Administration&gt;csadm GenRandom=64  aa8caa7c2880361153cd7987badc8041 1d5b5803e9055ec2d087fb188f964808 0cf5eb8d867dec296158dbb3756bb63b 4491480d9b0529b9455199625ccd9a21</pre>

## 4.5 Commands for Managing the User Authentication Keys

In this chapter the csadm commands for generation, administration and backup of user authentication keys (or tests keys) are described. The generated keys may be stored either on a smartcard or in a keyfile.

The following command group contains internal functions of csadm. No connection to a device will be established.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

### 4.5.1 GenKey

With the `GenKey` command, you can generate an RSA or ECDSA key pair. The keys can either be generated and saved directly on a smartcard (recommended) or they can be stored in an encrypted or unencrypted keyfile.





Only use an unencrypted keyfile for testing purposes if security is not relevant.

- **Key generation and storage on smartcard (recommended):**

Your keys are generated on the smartcard and stored there. All operations with the private key are carried out on the card. The private key never leaves the card. Users need a PIN to access the smartcard.



The Java smartcard version must be 2.0.0.0 or later. If you want to store RSA keys longer than 2048 bits on the smartcard, Utimaco smartcard version 3.0.0.0 or higher is required. Use the `csadm GetCardInfo` command to check the version: If no output line `Utimaco Smartcard version` is shown, the smartcard does not support generating keys on the smartcard and storing them.

These keys cannot leave the smartcard and cannot be backed up.

**Fallback users**

Since these keys cannot be backed up, the owner must be issued with at least a second smartcard containing an authentication key that is assigned to a so-called "fallback user." This smartcard then can be used if the initial one is lost or broken.

For the additional smartcards, new users with the appropriate rights (fallback users) must be created and smartcards must be issued for them (for example `James`, `James_1`, ...).

This means that the person has then several smartcards which they can use for authentication. If the initial smartcard becomes unusable for any reason, the owner can log in as a fallback user using the associated smartcards.

The smartcards for the fallback users must be stored in a safe place!

- **Keys stored in keyfile - the keyfile stored on a smartcard:**

A keyfile `*.key` containing the keys is generated. You can then store the keyfile on a smartcard. For storing keys on smartcards see the `csadm SaveKey` command.

For backup purposes, you can store the keyfiles on several smartcards. The smartcards and the associated PINs must be stored in a safe place - if necessary, store these separate from one another.


- **Keys stored in a keyfile:**

A keyfile `*.key` containing the keys is generated. The generated keyfile can be secured with a passphrase or created unencrypted (not recommended).

Where the keys are generated/stored is specified by a key specifier, see *Storage and Specification of RSA and ECDSA Keys for Authentication*.

The `GenKey` command is executed locally without a connection to the cHSM. Therefore, the state or mode of any underlying cHSM is irrelevant to the command execution, and no authentication to the cHSM is necessary.

<b>Syntax</b>	<code>csadm KeyType=[RSA EC] Key=&lt;keyspec&gt;[,&lt;bitlength&gt;],&lt;owner&gt;</code>
---------------	---

Parameter	Description
<keytype>	Type of the key: <ul style="list-style-type: none"> <li>▪ <code>RSA</code> (default)</li> <li>▪ <code>EC</code> for ECDSA</li> </ul>
<keyspec>	<p>The key specifier determines where the user authentication keys are stored/generated (smartcard or keyfile).</p> <ul style="list-style-type: none"> <li>▪ Smartcard specifier, for example, <code>:cs2:cjo:USB0</code> The user authentication keys are generated on the smartcard and are stored on it. They cannot leave this smartcard and cannot be backed up.</li> </ul> <hr/> <div style="display: flex; align-items: center;">  <div style="margin-left: 10px;"> <p>A newly generated RSA key overwrites an already existing RSA key on the smartcard. A newly generated EC key overwrites an already existing EC key on the smartcard.</p> </div> </div> <hr/> <ul style="list-style-type: none"> <li>▪ Storage location and name of the keyfile ( <code>*.key</code> ) If you want to create an encrypted keyfile you have to add a passphrase: <code>... \*.key [#&lt;passphrase&gt;]</code> where <code>&lt;passphrase&gt;</code> is the passphrase used for protecting the keyfile. If you do not specify a passphrase, a plaintext keyfile is generated (not recommended). The passphrase can be specified in 2 ways: <ul style="list-style-type: none"> <li>• in plain text For example: <code>MyKey.key#mypwd</code></li> <li>• by adding the <code>#ask</code> string that triggers a hidden passphrase entry (highly recommended). You will be asked for the passphrase and have to confirm it. The passphrase will not be displayed when you enter it. For example: <code>... \MyKey.key#ask</code></li> </ul> </li> </ul>
<bitlength>	Bit length of the generated RSA key, $512 \leq \text{bitlength} \leq 8192$ . Default value: <code>2048</code> bit.

Parameter	Description
<curve>	Name of the elliptic curve if the <KeyType> parameter is set to EC. Default value: brainpoolP320t1. If the <keyspec> parameter is set to a smartcard specifier, only the brainpoolP320t1 value is supported for the <curve> parameter.
<owner>	Owner or name of the key

<b>Example</b>	<ul style="list-style-type: none"> <li>Generate RSA keys on the smartcard csadm  <code>csadm GenKey=:cs2:cjo:USB0,"cHSM User"</code></li> <li>Generates an encrypted keyfile that contains RSA keys. You will be asked for the passphrase.  <code>csadm GenKey=C:\my_keys\testkey1.key#ask,2048,TestUser</code></li> </ul>
----------------	--

<b>Output</b>	If the command executes successfully, there is no output.
---------------	---

## 4.5.2 SaveKey

This command reads a key from one storage medium and stores it to another storage medium. The key may either be the public or private part of a key pair or one half of a backup key set.

The following table shows the possible combinations of source and target mediums:

Key Type	Source Medium	Target Medium
Public	keyfile	keyfile
	smartcard	keyfile
Private + Public	keyfile	keyfile or smartcard
Backup	smartcard	keyfile or smartcard

Table 16: Possible source and target medium combinations



It is neither possible to transfer the private key part out of a smartcard to any other medium, nor to transfer only a public key part to a smartcard.

This command is executed locally without any connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

<b>Syntax</b>	For a public key:
	<code>csadm [KeyType=&lt;keytype&gt;] PubKey=&lt;source&gt; SaveKey=&lt;target&gt;</code>
	For a private key:
	<code>csadm [KeyType=&lt;keytype&gt;] PrvKey=&lt;source&gt; SaveKey=&lt;target&gt;</code>
	For a backup key:
	<code>csadm [KeyType=&lt;keytype&gt;] BckKey=&lt;source&gt; SaveKey=&lt;target&gt;</code>

<b>Parameter</b>	<b>Description</b>
<keytype>	Key type: RSA (default) or EC
<source>	<p>Filename or key specifier where the key should be read from:</p> <ul style="list-style-type: none"> <li>For a public key: smartcard specifier or keyfile</li> <li>For private key: keyfile <ul style="list-style-type: none"> <li>In case of an encrypted keyfile, the password has to be appended after the filename, separated by a '#'. If no password is given (or the password is 'ask'), hidden password entry is performed.</li> <li>For backup key: Smartcard specifier where the two XOR parts of the backup key are read from two smartcards</li> </ul> </li> </ul>
<target>	<p>Filename or key specifier defining where the key should be stored:</p> <ul style="list-style-type: none"> <li>For public key: keyfile</li> <li>For private key or backup key: smartcard specifier or keyfile</li> </ul> <p>In case of an encrypted keyfile, the password has to be appended after the filename, separated by a '#'. If the password is given as 'ask', hidden password entry is performed.</p>

**Example**

- Read an RSA public key from a smartcard and store in a keyfile  
`csadm PubKey=:cs2:cjo:USB0 SaveKey=C:\keys\pubkey1.key`
- Read an ECDSA private key from an encrypted keyfile and store it on a smartcard  
`csadm KeyType=EC PrvKey=C:\keys\prvECkey1.key#ask  
SaveKey=:cs2:cjo:USB0`
- Read an ECDSA private key from a keyfile and store it in an encrypted keyfile  
`csadm KeyType=EC PrvKey=C:\keys\prvECkey1.key SaveKey=C:\keys\prvECkey1_enc.key#ask`
- Read an ECDSA backup key from two smartcards (two XOR parts) and store it on another smartcard  
`csadm KeyType=EC BckKey=:cs2:cjo:USB0  
SaveKey=:cs2:cjo:USB0`
- Read an RSA backup key from two smartcards (two XOR parts) and store it in an encrypted keyfile  
`csadm BckKey=:cs2:cjo:USB0 SaveKey=C:\keys\RSAPubkey1_enc.key#ask`

**Output**

Upon successful execution of the command, no output is given.

If smartcards are used, a PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.

Watch the display of the PIN pad for instructions on further command processing.

### 4.5.3 BackupKey

This command reads a key from a keyfile, splits it into two XOR parts and saves the parts on two smartcards for backup matters.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

If smartcards are used, a PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.

Watch the display of the PIN pad for instructions on further command processing.



A backup generated by the `csadm BackupKey` command is stored on smartcards. An MBK rollover is irrelevant for this backup. This backup can be restored after an MBK rollover.

<b>Syntax</b>	<code>csadm [KeyType=&lt;keytype&gt;] PrvKey=&lt;keyfile&gt; BackupKey=&lt;cardspec&gt;</code>
---------------	--

Parameter	Description
<keytype>	Key type: RSA (default ) or EC
<keyfile>	File where the key is stored ( *.key )
<cardspec>	Smartcard specifier of the back-up smartcards

<b>Example</b>	<code>csadm PrvKey=C:\my_keys\prvkey1.key BackupKey=:cs2:cjo:USB0 csadm KeyType=EC PrvKey=C:\my_keys\myprvkey.key#mypassword BackupKey=:cs2:cjo:USB0</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

#### 4.5.4 CopyBackupCard

This command copies one XOR-half of a key backup (backup of user authentication key) from one smartcard to another.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

<b>Syntax</b>	<code>csadm [KeyType=&lt;keytype&gt;] CopyBackupCard=&lt;keyspec&gt;</code>
---------------	---

Parameter	Description
<keytype>	Key type. RSA (default) or EC. The <code>csadm CopyBackupCard</code> command copies an RSA share or an EC share, which is specified by the <code>&lt; keytype &gt;</code> parameter. The corresponding action initiated by Java-based GUI CryptoServer Administration Tool (CAT) copies an RSA share and an EC share at the same time.
<keyspec>	Key specifier of the backup smartcards, source and target

<b>Example</b>	<code>csadm KeyType=EC CopyBackupCard=:cs2:cjo:USB0</code>
----------------	--

**Output**

Upon successful execution of the command, no output is given.



A PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.

Watch the display of the PIN pad for instructions on further command processing.

### 4.5.5 GetCardInfo

This command reads information about keys eventually stored on a smartcard. The info records of the user authentication key (e.g., Default Administrator Key) and key backup share are scanned and output.

A smartcard can contain an RSA key, an elliptic-curve cryptography key, an RSA key backup share, an elliptic key backup share and an MBK backup share at the same time.

- RSA-Key

An RSA key. Use the `csadm GenKey` command or CAT to generate it. The default value is Utimaco IS GmbH / Init-Dev-1-Key or Utimaco IS GmbH / ADMIN-Key (older default value). Both default keys are the key for the default administrator (ADMIN user name).

- ECC-Key

An elliptic-curve cryptography key. Use the `csadm GenKey` command or CAT to generate it. The default value is Utimaco IS GmbH / Default\_EC\_Key.

- RSA-Backup

An RSA key backup share on the smartcard. It cannot be generated using csadm. Use CAT instead.

- ECC-Backup

An elliptic-curve cryptography key backup share on the smartcard. It cannot be generated using csadm. Use CAT instead.

- MBK

A backup share of a master backup key (MBK). Use the `csadm MBKGenerateKey` command or CAT to generate it. If you want to retrieve information only about MBK shares consider to perform the `csadm MBKCardInfo` instead of performing `csadm GetCardInfo`.

The `GetCardInfo` command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

<b>Syntax</b>	<code>csadm GetCardInfo=&lt;keyspec&gt;</code>
---------------	--

Parameter	Description
<keyspec>	Key specifier of the backup-smartcards, source and target

<b>Example</b>	<code>csadm GetCardInfo=:cs2:cjo:USB0</code>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the card information is returned.</p> <pre> Utimaco Smartcard version: 2.2.0.0 RSA-Key:      Utimaco IS GmbH / Init-Dev-1-Key ECC-Key:      Utimaco IS GmbH / Default_EC_Key RSA-Backup:   RsaKey02 #2 ECC-Backup:   EcKey #2 MBK:          13: 32 6 AES MbkAes28 09.05.2019 09:10:15 02 02 37BAFABF49C90D0C 14: 32 3 AES MbkAes29 09.05.2019 09:04:11 00 00 8F21825F743C3A49 15: 32 6 AES test 06.03.2017 14:29:15 02 01 BC954D6AD6132A9C </pre> <p>The output line starting with Utimaco Smartcard version is not present if a Javacard with version &lt; 1.0.3.0 or a TC30 card is used. The <code>csadm GetCardInfo</code> command is the only way to retrieve the Javacard version</p>
---------------	--



A PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.

Watch the display of the PIN pad for instructions on further command processing.

### 4.5.6 ChangePassword

This command changes the password of an encrypted keyfile (`*.key`).



This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

This command can also be used to convert a plaintext keyfile into an encrypted keyfile and vice versa:

- If the command shall be used to encrypt a keyfile that has been a plaintext keyfile before, the parameter `<old_password>` has to be omitted and only the `NewPassword=<new_password>` parameter has to be given.
- If the command shall be used to decrypt a keyfile, i.e., to turn an encrypted keyfile into a plaintext keyfile the parameter `NewPassword=<new_password>`

<b>Syntax</b>	<code>csadm [KeyType=&lt;keytype&gt;] NewPassword=&lt;new_password&gt; ChangePassword=&lt;keyfile&gt;[#old_password]</code>
---------------	---

Parameter	Description
<code>&lt;keytype&gt;</code>	Type of the key, either RSA (default) or EC
<code>&lt;oldpassword&gt;</code>	Old password of the encrypted keyfile <ul style="list-style-type: none"> <li>▪ If no <code>old_password</code> is given (or <code>old_password</code> is 'ask'), hidden password entry is possible, which we strongly recommend</li> <li>▪ old password in plaintext</li> </ul>
<code>&lt;newpassword&gt;</code>	New password for the encrypted keyfile <ul style="list-style-type: none"> <li>▪ string ask that defines that hidden password entry is used, which we strongly recommend</li> <li>▪ new password in plaintext</li> </ul>
<code>&lt;keyfile&gt;</code>	Encrypted keyfile (*.key)

<b>Example</b>	<ul style="list-style-type: none"> <li>▪ Generate an encrypted keyfile containing an RSA key pair <code>csadm GenKey=C:\my_keys\testkey1.key#ask,2048,TestKey</code></li> <li>▪ Generate an encrypted keyfile containing an ECDSA key pair <code>csadm KeyType=EC GenKey=C:\my_keys\testkey2.key#123456,secp256k1,ECkey</code></li> <li>▪ Generate an ECDSA key pair on a smartcard <code>csadm KeyType=EC GenKey=:cs2:cjo:USB0,"CryptoServer User"</code></li> </ul>
----------------	---

**Output**

Upon successful execution of the command, no output is given.



When the credentials of a user are changed and the new credentials are equal to the old credentials, then the system returns an error and displays the message: "The specified user credentials are already in use by the specified user."

### 4.5.7 ChangePIN

This command changes the PIN of a given smartcard. A PIN pad must be used for this command. Only PINs consisting of 6 up to 12 digits are supported.



The PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer. Follow the instructions on the display of the PIN pad for further processing.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

**Syntax**

```
csadm ChangePIN=<keyspec>
```

**Parameter****Description**

<keyspec>

Specifier for smartcard type, reader type and USB port

**Example**

```
csadm ChangePIN=:cs2:cjo:USB0
```

**Output**

Upon successful execution of the command, no output is given.

### 4.5.8 BackupDatabase

This command creates a backup of all entries of the given device database (for example, the cryptographic key database ( `CXIKEY.db` ) or the audit log signature key database ( `auditkey.db` )).

The function creates a file with the name of the database and stores it in the current directory of the command line. Each database entry in the created backup file is encrypted with the Master Backup Key (MBK) of the device. Due to a check value (MAC calculated with the MBK) over each database record it is not possible to change any item (for example, database index, key record).

The database for which the backup shall be created must have the file extension `*.db`. The function creates a file with ASCII characters and additional information about the used MBK. If a file with the name of the database already exists in the current directory, it will be overwritten. It is not possible to backup the MBK database or the database with the secure messaging session keys.



Perform the `csadm MBKListKeys` command to determine which Master Backup Key (MBK) is currently in use in MBK slot 3 by the device. This MBK is used by the `csadm BackupDatabase` command to protect the backup file to be generated. If the MBK in MBK slot 3 is the autogenerated MBK named `AUTO-GEN`, the `csadm BackupDatabase` command cannot be performed. Import a different MBK into MBK slot 3 using the `csadm MBKImportKey` command.

It is important to note down which MBK has been used because for a successful restoring of this backup file at a later date it is necessary that the same MBK is in MBK slot 3. Otherwise, for example, after the execution of a `csadm MBKImportKey` command or after an MBK rollover, the backup file is inaccessible.



Only one database backup can be saved at a time. To back up several databases, the command must be executed separately for each individual database.

As of SecurityServer 6.0.0, a new database backup structure/format has been introduced.

- For creating and restoring backups, it is recommended to use a csadm version that matches the firmware version.
- Backups with the format earlier than 6.0.0 are still possible to be restored, except in FIPS mode because FIPS does not allow the old format.

- Restoring a backup with the new format into firmware version < 6.0.0 is not possible. Generally, restoring backups of a newer firmware version into older firmware is not supported.
- As of version 6.0.0, customers must use the `csadm BackupUser` command to back up users, because `csadm ... BackupDatabase=user.db` is not supported anymore.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a `csadm` command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

The following is an example for a correct `csadm` command entry in the Microsoft PowerShell:

```
csadm [Dev=<device>] LogonSign="<user>,<keyspec>" <command>
```

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; BackupDatabase=&lt;name of database&gt;</code>
<b>Authentication</b>	Permission level 2 in user group 6 and 7 (22000000)
Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<name of database>	Name of the device database
<b>Example</b>	Back up the cryptographic key database <code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 BackupDatabase=CXIKEY.db</code>
<b>Output</b>	Upon successful execution of the command, no output is given.

## 4.5.9 RestoreDatabase

This command restores a database on the device from a backup file that was created with the `csadm BackupDatabase` command.

For the restore procedure, the function takes each record from the backup file, decrypts it inside the device with the Master Backup Key (MBK), verifies the MAC and stores it under the given database index. If an entry with the given database index already exists, it will be overwritten. The backup file must have the file extension `*.db`. It is not possible to restore the MBK database or the database with the secure messaging session keys.

As of SecurityServer 6.0.0, a new database backup structure/format has been introduced.

- For creating and restoring backups, it is recommended to use a csadm version that matches the firmware version.
- Backups with the format earlier than 6.0.0 are still possible to be restored, except in FIPS mode because FIPS does not allow the old format.
- Restoring a backup with the new format into firmware version < 6.0.0 is not possible. Generally, restoring backups of a newer firmware version into older firmware is not supported.
- As of version 6.0.0, customers must use the `csadm BackupUser` command to back up users, because `csadm ... BackupDatabase=user.db` is not supported anymore.



As of SecurityServer FIPS 140-3 6.0.0, a new firmware and a new backup format are applied. As a consequence, if you have backed up a database using a SecurityServer < 6.0.0 (i.e. using the old firmware with the old backup format), this database backup cannot be restored in one step by SecurityServer FIPS 140-3 >= 6.0.0.

Instead, proceed as follows:

1. Restore the database backup using SecurityServer non-FIPS >= 6.0.0.
2. Create a new database backup using SecurityServer non-FIPS >= 6.0.0 (i.e. using the new firmware with the new backup format).
3. Restore the new database backup using SecurityServer FIPS 140-3 >= 6.0.0 (i.e. using the new firmware with the new backup format).



We recommend performing a `csadm Restart` after executing the `RestoreDatabase` command:

If CXI configuration items are stored in the restored database, they are applied immediately to the CryptoServer. Other restored configuration items will not become effective until a CryptoServer restart will have been performed.



The same Master Backup Key (MBK) which has been used to create the backup file has to be stored on the target device.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

The following is an example for a correct csadm command entry in the Microsoft PowerShell:

```
csadm [Dev=<device>] LogonSign="<user>,<keyspec>" <command>
```

#### Syntax

```
csadm [Dev=<device>] <Authentication> ...  
RestoreDatabase=<backup_file>
```

#### Authentication

Permission level 2 in user group 6 and 7 (22000000)

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<backup_file>	Name of the backup database file (eventually with path)

#### Example

```
Restore the cryptographic key database  
csadm LogonSign=ADMIN,:cs2:cjo:USB0  
RestoreDatabase=CXIKEY.db
```

#### Output

Upon successful execution of the command, no output is given.

## 4.6 Commands for User Management

The device provides a sophisticated user management in order to maintain different users with different authentication mechanisms and permissions.

User's credentials are stored in the user database `user.db`, which is hosted on the device.

The respective commands to create, change or delete users are described in this chapter. Within this user management, the following properties can be assigned to each user:

<i><b>Property</b></i>	<i><b>Description</b></i>
Name	Username, up to 255 printable characters (must not contain a '~')

Property	Description																					
Permission	<p>A user's permission, for example, 2A000000, is an eight-digit sequence of hexadecimal values between 0 and F (15). Each digit stands for one of the eight device specific user groups. Each user group is assigned a specific group of functions/commands, and is uniquely identified by a number in the range from 0 to 7. User's permission in the user group 7 is displayed as the most left one, and the permission in user group 0 as the most right one. This means, user's permission in a specific user group defines the rights a user is granted on creation, allowing him to execute security-relevant functions on the device that always require user authentication.</p> <p>The value 0 means that the user has no rights in that particular user group.</p> <ul style="list-style-type: none"><li>▪ 1 means two-person rule: although the user can authenticate commands to the device, a second user is also required to do this.</li><li>▪ 2 means that the user is entitled to authenticate commands on his own.</li></ul> <p>Values 3 to F mean that the user can authenticate commands on his own as well as can participate in performing the n-person rule authentication (<math>n &gt; 2</math>), and M-of-N users authentication (where N is the number of all device users with the same role-based user profile, M is the number of users with the same role-based user profile required to authenticate specific command/function). 2 is the highest possible permission required by the device functions/commands by default. 3 to F might be required by functions with custom permissions defined in a signed configuration file <code>cmds.scf</code>.</p> <p>Some user groups are reserved by Utimaco for applications and their corresponding role-based user profiles as shown in the next table.</p> <table><tr><th>User Group</th><th>Application</th><th>Role-based user profile and permissions</th></tr><tr><td>0</td><td>All cryptographic interfaces</td><td>Cryptographic User und PKCS#11 User; 00000002</td></tr><tr><td>1</td><td>PKCS#11</td><td>PKCS#11 Key Manager; 00000020</td></tr><tr><td>2</td><td>PKCS#11</td><td>PKCS#11 Security Officer (SO); 00000200</td></tr><tr><td>5</td><td>NTP Administration</td><td>NTP Manager; 00200000</td></tr><tr><td>6</td><td>System Administration</td><td>System Manager; 02000000</td></tr><tr><td>7</td><td>User Management</td><td>User Manager; 20000000</td></tr></table> <p>All other user groups (3 and 4) can be used for customer-specific applications.</p>	User Group	Application	Role-based user profile and permissions	0	All cryptographic interfaces	Cryptographic User und PKCS#11 User; 00000002	1	PKCS#11	PKCS#11 Key Manager; 00000020	2	PKCS#11	PKCS#11 Security Officer (SO); 00000200	5	NTP Administration	NTP Manager; 00200000	6	System Administration	System Manager; 02000000	7	User Management	User Manager; 20000000
	User Group	Application	Role-based user profile and permissions																			
0	All cryptographic interfaces	Cryptographic User und PKCS#11 User; 00000002																				
1	PKCS#11	PKCS#11 Key Manager; 00000020																				
2	PKCS#11	PKCS#11 Security Officer (SO); 00000200																				
5	NTP Administration	NTP Manager; 00200000																				
6	System Administration	System Manager; 02000000																				
7	User Management	User Manager; 20000000																				
Mechanism	<ul style="list-style-type: none"><li>▪ RSA signature authentication mechanism</li><li>▪ ECDSA signature authentication mechanism</li><li>▪ HMAC password authentication mechanism</li></ul>																					



Property	Description
Attributes	<p>A string containing an assignment:  <code>&lt;AttrName&gt;=&lt;AttrValue&gt;</code>  Examples:</p> <ul style="list-style-type: none"> <li><code>CXI_GROUP=msk*</code> or <code>CXI_GROUP=*_mbk</code>  This attribute defines that the cryptographic users they have been assigned to are permitted to generate, use, delete, export and import keys belonging to the CXI groups msk_1, msk_2, msk3 and so on, or to the groups A_mbk, B_mbk and so on.  A concatenation of two attributes, e.g.,  <code>CXI_GROUP=msk*, CXI_GROUP=*_mbk</code>  is not supported.</li> <li><code>CXI_GROUP=SLOT_XXXX</code> This attribute must be set for PKCS#11 users of PKCS#11 slot X</li> </ul> <p>The meaning of the Attributes depends on the application loaded into the device.  The permission of a user may be restricted to a group of objects (keys, configuration and storage objects) that matches the user attribute <code>CXI_GROUP</code>. This means, a user is only allowed to access keys within his key group, and has no access to keys from other groups. If the user group attribute contains wildcards ('?' or '*'), the user may access multiple key groups; as example 2 above shows, <code>CXI_GROUP=msk*</code> allows the user to access msk01 as well as msk02.  If a user was created without a user group attribute, he is only allowed to access global objects that do not belong to any group of objects.  If a user was created with the attribute <code>CXI_GROUP=*</code>, he is allowed to access all objects regardless of their group property.  The defined value is assigned to the user as the attribute <code>A[]</code>, e.g.  <code>A[CXI_GROUP=SLOT_0000]</code>, and can be displayed by executing the <code>csadm ListUser</code> command.</p>
HashAlgo	<p>Hash algorithm to be used for RSA signature authentication, ECDSA signature authentication or HMAC password authentication mechanism in case that not the default hash algorithm shall be used for this user, which is SHA-256 for RSA signature authentication, ECDSA signature authentication and HMAC password authentication.  The value can be one of the following algorithms:</p> <ul style="list-style-type: none"> <li>SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512</li> <li>MD5</li> <li>RIPEMD-160</li> </ul> <p>The defined value is assigned to the user as the attribute <code>H[]</code>, for example, <code>H[SHA-256]</code>.  In FIPS mode, only the default hash algorithms are supported.</p>

The creation of new users also requires an authentication:

- In general, authentication by one or two users with the permission to manage users (i.e., authentication level 2 in user group 7) is required.

- If a user with administrative rights shall be created: additionally, the authentication of one user with the permission to administrate the device is required (i.e., authentication level 2 in user group 7 and level 1 in user group 6).

For this reason, one initial user ADMIN is preconfigured and uses the RSA-Signature authentication mechanism with the Initialization Key. ADMIN has the exclusive permission of user management and administration (22000000) and does not require a second user to authenticate administrative commands.

The user ADMIN may be deleted from the user database only if one or more other users have been created, who – alone or in combination – possess the minimum permission to create users, including a user with administrative rights.



The user management of the device checks if the sum of the permissions of the remaining users with signature authentication mechanism (RSA, ECDSA) still allows user management and the creation of a user with administrative rights (i.e., resulting permission  $\geq 21000000$ ) and denies the deletion of a user eventually.

By this means users can never- accidentally - lock out themselves from the device.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

### 4.6.1 ListUser

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

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] ListUser</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;user&gt;</code>	Existing user name

**Example**

csadm Dev=192.168.1.1 ListUser

**Output**

Upon successful execution of the command, a list of all existing users is given. The initial user ADMIN is always displayed even if the database `user.db` is not yet present.

Name	Permission	Mechanism	Attributes
ADMIN	22000000	RSA Sign	
Z[[0]I[0]H[SHA-256]			
sm	00000022	HMAC passwd	Z[1]I[0]
Sven	22000022	HMAC passwd	I[0]
SO_0007	00000201	HMAC passwd	
A[CXI_GROUP=SLOT_0007]L[0007]I[0]			
CryptU	00000002	RSA sign	A[CXI_GROUP=msk]

Parameter	Description
A[]	Application-depending attributes, for example, A[CXI_GROUP=SLOT_0007]
H[]	Non-default hash algorithms used for the authentication mechanism of that user, for example, H[SHA-256]. The H[] attribute is only shown if this attribute has been explicitly set when the user has been added to the device.
Z[]	Counter for the number of consecutively failed authentication attempts marked by the tag Z, e.g. Z[1]. The attribute Z is set for every user on first authentication attempt to the default Z[0]. If the authentication succeeded, Z remains Z[0], otherwise the counter is incremented by one.
L[]	PKCS#11 slot label which is assigned to the slot's Security Officers (SO) during slot initialization. By default, this attribute is set to CryptoServer PKCS11 Token. It can be changed by the default device Administrator ADMIN or a user administrator during PKCS#11 slot initialization using one of the PKCS#11 administration tools – P11CAT or p11tool2. The PKCS#11 slot must not be confused with the MBK slot.

Parameter	Description
I[]	<p>I[] only applies to the ADMIN user and to users with HMAC password authentication.</p> <ul style="list-style-type: none"> <li>▪ ADMIN user <ul style="list-style-type: none"> <li>• I[1] <p>Initial state of the ADMIN user at delivery (except for an ADMIN user on a CryptoServer Simulator). The credentials of the ADMIN user are unchanged, i.e., the ADMIN.key file is used. In this state, the ADMIN user cannot perform commands he/she needs an authentication for, except for the csadm ChangeUser command. To enable the ADMIN user to perform commands he/she needs an authentication for, the ADMIN user must change the credentials by performing the csadm ChangeUser command.</p> </li> <li>• I[0] <p>The ADMIN user can perform commands he/she needs an authentication for, i.e., one of the following cases applies:</p> <ul style="list-style-type: none"> <li>• Initial state of the ADMIN user on a CryptoServer Simulator at delivery, i.e., the ADMIN_SIM.key file is used. The ADMIN_SIM.key file is available in the (default) C:\Program Files\Utimaco\SecurityServer\Administration directory of the CryptoServer installation. In the product bundle, the ADMIN_SIM.key file is available in the Software\Windows\Administration\key directory.</li> <li>• The ADMIN user has changed his/her credentials.</li> </ul> </li> </ul> </li> <li>▪ User with HMAC password authentication <ul style="list-style-type: none"> <li>• I[1] <p>The user cannot perform commands he/she needs an authentication for (except for the csadm ChangeUser command), i.e., one of the following cases applies:</p> <ul style="list-style-type: none"> <li>• Initial state when the user has been created.</li> </ul> </li> </ul> </li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>Another user (e.g., an administrator) has changed the user's password.</li> </ul> <p>To enable the user to perform commands he/she needs an authentication for, the user must change his/her credentials. To do so, this user must perform the <code>csadm ChangeUser</code> command. If the user is a PKCS#11 Security Officer (SO) or a PKCS#11 normal user, he/she may perform the <code>p11tool2 SetPIN</code> command as an alternative. It is important that the changes are performed by the user himself/herself and not, for example, by an administrator.</p> <ul style="list-style-type: none"> <li><code>I[0]</code> The user has changed his/her own password. The user can perform commands he/she needs an authentication for.</li> </ul> <p>If user data is backed up using an old firmware not supporting the <code>I[]</code> attribute and this user data is restored using a new firmware supporting the <code>I[]</code> attribute, the restored users do not have the <code>I[]</code> attribute, i.e., the restored users do not need to change their credentials.</p> <p>If users having the <code>I[]</code> attribute are backed up and restored, the restored users have the <code>I[]</code> attribute in the state they had before the backup. You cannot downgrade the <code>I[]</code> attribute, i.e., perform a <code>csadm BackupUser</code> command for a user having the <code>I[]</code> attribute and then perform a <code>csadm RestoreUser</code> command on an old firmware not supporting the <code>I[]</code> attribute. This procedure would cause a <code>Bad user attribute</code> error (B0830017).</p>

Table 17: Meaning of the return parameters of csadm ListUser




The attribute L is only set when the user with role SO is created during slot initialization with one of the PKCS#11 administration tools – P11CAT or p11tool2. It cannot be set in case you create the SO for a specific PKCS#11 slot with CAT or csadm.

## 4.6.2 AddUser

With this command a new user is added to the user database of the device.

<b>Syntax</b>	<pre>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; ... AddUser=&lt;user&gt;,&lt;permission&gt;,&lt;mechanism&gt;,&lt;auth_token&gt;</pre>
---------------	---

<b>Authentication</b>	<p>The command must be authenticated with permission 2 in user group 7 (20000000)</p> <hr/> <div style="display: flex; align-items: center;">  <p>If a user with permission for administrating the device is created (i.e., a user with permission 1 in user group 6), the command must be authenticated with permission 2 in the user group 7 and permission 1 in the user group 6.</p> </div> <hr/>
-----------------------	--

<b>Parameter</b>	<b>Description</b>
<device>	<p>Device address</p> <p>This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.</p>
<user>	<p>The new user's name</p> <p>If you want to create a PKCS#11 Security Officer (SO), the name of a Security Officer must be <code>SO_xxxx</code> with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from <code>SO_0000</code> to <code>SO_9999</code>.</p> <p>If you want to create a PKCS#11 User, the name of the User must be <code>USR_xxxx</code> with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from <code>USR_0000</code> to <code>USR_9999</code>.</p> <p>If you want to create a PKCS#11 key manager, it is advisable but not mandatory to use <code>KM_xxxx</code> as the name of the key manager with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from <code>KM_0000</code> to <code>KM_9999</code>.</p>
<permission>	<p>The new user's permissions:</p> <p>Eight digits, <code>xxxxxxxx</code>, each representing a permission in a specific user group. Leading zeros may be omitted.</p> <p>A list of attributes, put into curly braces {}, can be appended to the permission digits, see the example below.</p> <p>If you want to create a PKCS#11 Security Officer (SO), assign <code>00000200{CXI_GROUP=SLOT_xxxx}</code> or <code>{CXI_GROUP=SLOT_xxxx}00000200</code> with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The slot specifier may range from <code>SLOT_0000</code> to <code>SLOT_9999</code>.</p> <p>If you want to create the PKCS#11 User, assign <code>00000002{CXI_GROUP=SLOT_xxxx}</code> or <code>{CXI_GROUP=SLOT_xxxx}00000002</code> with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The slot specifier may range from <code>SLOT_0000</code> to <code>SLOT_9999</code>.</p> <p>If you want to create a PKCS#11 Key Manager, assign <code>00000020{CXI_GROUP=SLOT_xxxx}</code> or <code>{CXI_GROUP=SLOT_xxxx}00000020</code> with <code>xxxx</code> being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The slot specifier may range from <code>SLOT_0000</code> to <code>SLOT_9999</code>.</p>

Parameter	Description
<mechanism>	<p>The new user's authentication mechanism</p> <p>During execution of the <code>csadm AddUser</code> command the public part of the RSA or ECDSA key is read from the smartcard or keyfile and stored in the user database of the device.</p> <p>For a PKCS#11 Security Officer or the PKCS#11 User, only HMAC password authentication is supported.</p> <ul style="list-style-type: none"> <li>▪ <code>rsasign</code>            RSA signature authentication.            Once created, a user uses either a keyfile or a smartcard in a PIN pad that is locally or remotely connected to the USB port of the computer csadm is running on.            If the new user's private RSA key is available on a smartcard, this smartcard must be inserted into a PIN pad that is connected to the computer csadm is running on during the execution of the <code>csadm AddUser</code> command.</li> <li>▪ <code>ecdsa</code>            ECDSA signature authentication.            Once created, a user uses either a keyfile or a smartcard in a PIN pad that is locally or remotely connected to the USB port of the computer csadm is running on.            If the new user's private ECDSA key is available on a smartcard, this smartcard must be inserted into a PIN pad that is connected to the computer csadm is running on during the execution of the <code>csadm AddUser</code> command.</li> <li>▪ <code>rsasc</code>            RSA smartcard authentication.            Once created, a user uses a smartcard in a PIN pad that is directly connected to the USB port of the PCIe card. It is not sufficient to connect the PIN pad to a port of the computer csadm is running on. The instructions for using a Local PIN Pad for a remote device do not apply here.            If the created user is used on a LAN device, the PIN pad may also be connected to a port on the front panel that is directly connected to the PCIe card. Depending on the LAN device version, this is a USB port labeled <b>CS USB</b>, <b>USB CS</b>, <b>USB CS2</b> or <b>HSM</b>.            However, if you use the created user on the device Simulator, the smartcard must be inserted into a PIN pad that is connected to the computer csadm is running on (USB port) or to another computer.            If the private RSA key of the new user is available on a smartcard, this smartcard must be inserted into a PIN pad that is connected to the computer csadm is running on (USB port) or to another computer during the execution of the <code>csadm AddUser</code> command.</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>▪ <code>hmacpwd</code> HMAC password authentication.</li> </ul> <p>Once created, a user uses an HMAC password-based key-derived function (HMAC-PBKDF) for authentication. The HMAC-PBKDF according to NIST SP 800-132 does not use the HMAC password itself for authentication but a function derived from the HMAC password. For HMAC-PBKDF, 1000 iterations are used here. This number of iterations, as used for the key derivation from a given password, is fix and not configurable. HMAC-PBKDF is only applied if on both sides, the host side and the firmware side, HMAC-PBKDF is applied. If it is not available on one of these sides, the legacy version of the HMAC password-based mechanism is applied, whereby the user's password is used directly as an HMAC key. In FIPS mode, using HMAC-PBKDF is mandatory.</p> <p>If a user with HMAC password authentication is created, this user cannot perform commands he/she needs an authentication for (except for the <code>csadm ChangeUser</code> command). The <code>csadm ListUser</code> command shows this user with the <code>I[1]</code> attribute value. To enable the user to perform commands he/she needs an authentication for, the user must change his/her credentials. To do so, this user must perform the <code>csadm ChangeUser</code> command. If the user is a PKCS#11 Security Officer (SO) or a PKCS#11 normal user, he/she may perform the <code>pl1tool2 SetPIN</code> command as an alternative. It is important that the changes are performed by the user himself/herself and not, for example, by an administrator. The <code>csadm ListUser</code> command then shows this user with the <code>I[0]</code> attribute value.</p>
<auth_token>	<p>Authentication token (key specifier or password) of the user to be created, depending on user's authentication mechanism:</p> <p>If the <code>&lt;mechanism&gt;</code> parameter has the value <code>rsasign</code>, <code>rsasc</code> or <code>ecdsa</code>, the <code>&lt;auth_token&gt;</code> parameter is the key specifier for the public part of the user's RSA key or ECDSA key.</p> <p>If the <code>&lt;mechanism&gt;</code> parameter has the value <code>rsasign</code>, <code>rsasc</code> or <code>ecdsa</code> and if <code>&lt;auth_token&gt;</code> specifies a smartcard, this smartcard must be inserted into a PIN pad that is connected to the computer where csadm is running (USB port) or to another computer during the execution of the <code>csadm AddUser</code> command. This applies as well to the device Simulator.</p> <p>In case of <code>rsasign</code>, <code>rsasc</code> or <code>ecdsa</code>, a name of a hash algorithm, put into curly brackets {}, can be appended or prepended to the key specifier. For FIPS, only the default hash algorithms are supported.</p> <p>If the <code>&lt;mechanism&gt;</code> parameter has the value <code>hmacpwd</code>, the <code>&lt;auth_token&gt;</code> parameter is the new user's password (length between 8 and 1024 characters). A name of a hash algorithm, put into curly braces {}, can be appended or prepended to the password. For FIPS, only the default hash algorithms are supported. For hidden password entry the string 'ask' is used as a placeholder. We strongly recommend using a hidden password entry.</p> <p>If a 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.</p>



**Example**

- Add a user with RSA signature authentication  
`csadm LogonSign=ADMIN,:cs2:cjo:USB0`  
`AddUser=Sven,01000000,rsasign,{SHA-256}key1.key`  
 This command is not supported for FIPS because of the non-default hash algorithm SHA-256.  
 It is quite useful to append a `ListUser` command to verify the result of the user creation.  
`csadm LogonSign=ADMIN,:cs2:cjo:USB0`  
`AddUser=Sven,01000000,rsasign,{SHA-256}key1.key`  
`ListUser`
- Add a user with RSA smartcard authentication  
`csadm LogonSign=ADMIN,:cs2:cjo:USB0`  
`AddUser=USR2,00000010,rsasc,:cs2:cjo:USB0`
- The user administrator AdminSc (22000000) uses RSA smartcard authentication and a Utimaco cyberJack one PIN pad directly connected to the PCIe card. He authenticates a `csadm AddUser` command to create a user `UsrRsaSc (00000002)` who is assigned to the 1234 cryptographic key group, uses RSA smartcard authentication and whose private key is stored on a smartcard inserted in a Utimaco cyberJack one PIN pad connected to the computer `csadm` is running on.  
`csadm Dev=PCI:0 LogonSign=AdminSc,:cs2:cjo:USB0`  
`AddUser=UsrRsaSc,00000002{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0`
- The same as the last example but without a smartcard specifier for the user administrator AdminSc who is automatically authenticated with the PIN 123456. The leading zeros of the user permission have been removed.  
`csadm Dev=PCI:0 LogonSign=AdminSc,#123456`  
`AddUser=UsrRsaSc,2{CXI_GROUP=1234},rsasc,:cs2:cjo:USB0`  
 The smartcard specifier of the user to be created must not be omitted.
- Add a user with ECDSA signature authentication  
`csadm LogonSign=ADMIN,:cs2:cjo:USB0`  
`AddUser=Sven,01000000,ecdsa,key1.key`
- Add a user with PKCS#11 Key Manager role in PKC#11 slot 4 and HMAC password authentication  
`csadm LogonSign=ADMIN,:cs2:cjo:USB0`  
`AddUser=KM_0004,020{CXI_GROUP=SLOT_0004},hmacpwd,ask`

**Output**

Upon successful execution of the command, no output is given.

If the authentication of the command `AddUser` requires a smartcard and the source of the new user's public authentication key is a smartcard too, follow the instructions on the display of the PIN pad. You have to insert the smartcard for the command authentication key first, and then the smartcard containing the new user authentication key. This situation occurs in the following scenarios:

- Real hardware device
  - Command authentication key: RSA or ECDSA signature authentication with a smartcard
  - New user authentication key: RSA or ECDSA signature authentication with a smartcard or RSA smartcard authentication
- Device Simulator
  - Command authentication key: RSA or ECDSA signature authentication with a smartcard or RSA smartcard authentication
  - New user authentication key: RSA or ECDSA signature authentication with a smartcard or RSA smartcard authentication

#### **4.6.2.1 Adding PKCS#11 Security Officers with RSA Signature Authentication with a Smartcard**

##### **Prerequisites**

- The name of a Security Officer must be `SO_XXXX` with `XXXX` being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from `SO_0000` to `SO_9999`.
- The Security Officer must have at least permission 00000200.
- The Security Officer must be assigned to the appropriate key group/PKCS#11 slot. The slot specifier may range from `SLOT_0000` to `SLOT_9999`.
- Only HMAC password authentication is supported.

However, if you want to enforce having two users with RSA signature authentication with a smartcard to perform actions as a Security Officer, there is a solution. This section describes how to create the needed users. In total, the following three users are needed:

- A Security Officer named `SO_0000` (0000 for PKCS#11 slot 0) with permission 00000000 and HMAC password authentication.  
The purpose of this user is to provide a user with the needed name `SO_0000` and the HMAC password authentication enforced by PKCS#11. The permission must be 00000000 so that this user cannot perform any action at all and other users providing

the needed permissions and the desired RSA signature authentications with a smartcard are needed.

- A Security Officer named, for example, SOSC1\_0000 with permission 00000100 and RSA signature authentication with a smartcard  
This user provides the desired RSA signature authentication with a smartcard and the first half of the needed permission.
- A Security Officer named, for example, SOSC2\_0000 with permission 00000100 and RSA signature authentication with a smartcard  
This user provides the desired RSA signature authentication with a smartcard and the second half of the needed permission.

If you log in all three users, the summarized permission of the three users is  $00000000 + 00000100 + 00000100 = 00000200$ .

Perform the following steps to create the needed users.

1. Create the Security Officer SO\_0000.

Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0  
AddUser=SO_0000,00000000{CXI_GROUP=SLOT_0000},hmacpwd,123456
```

2. Create the Security Officer SOSC1\_0000.

Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0  
AddUser=SOSC1_0000,00000100{CXI_GROUP=SLOT_0000},rsasign,:cs2:cjo:USB0
```

3. Create the Security Officer SOSC2\_0000.

Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0  
AddUser=SOSC2_0000,00000100{CXI_GROUP=SLOT_0000},rsasign,:cs2:cjo:USB0
```

4. Now you can perform a command as a Security Officer logging in the three users.

Example:

```
csadm LogonPass=SO_0000,123456 LogonSign=SOSC1_0000,:cs2:cjo:USB0  
LogonSign=SOSC2_0000,:cs2:cjo:USB0 ...
```

### 4.6.3 ChangeUser

With the command `ChangeUser` a user can change his/her own user authentication key or password, or a user with user management rights can change the authentication credentials of another user. Furthermore, the failed authentication counter of the user whose authentication credentials are to be changed is reset. If the user has been blocked for authentication because the number of failed authentication attempts has exceeded the maximum number of allowed failed authentication attempts, he/she is unblocked again.



A user is not allowed to change his authentication mechanism or permissions.

In case the user authentication key shall be changed, the user needs a new RSA or ECDSA user authentication key (on a smartcard or as a keyfile) as well as his/her old RSA or ECDSA user authentication key pair.

- In case of a user with RSA Signature or ECDSA Signature authentication mechanism:  
If the authentication of this command requires a smartcard and the source of the user's new public key is a smartcard, too, follow the instructions on the display of the PIN pad. You'll have to insert the smartcard for the command authentication (old key) first, and then the smartcard containing the user's new authentication key.
- In case that the authentication key of a user with a password-based authentication mechanism shall be changed, we strongly recommend using a hidden password entry.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; ChangeUser=&lt;user&gt;,&lt;auth_token&gt;</code>
---------------	---

<b>Authentication</b>	The command must be authenticated with permission 2 in user group 7 (20000000)
-----------------------	--

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<user>	Existing username

Parameter	Description
<auth_token>	User's new authentication token (key specifier or password), depending on the user's authentication mechanism: In the case of a user with a signature-based authentication mechanism: key specifier for public part of the user's new RSA or ECDSA key In the case of a user with a password-based authentication mechanism: the user's new password (minimum length 8 characters, for user with HMAC Password mechanism maximum length 1024 characters); for hidden password entry: string 'ask'. We strongly recommend using a hidden password entry.

<b>Example</b>	<ul style="list-style-type: none"> <li>▪ <code>csadm LogonSign=sven,:cs2:cjo:USB0 ChangeUser=sven,:cs2:cjo:USB0</code></li> <li>▪ <code>csadm LogonPass=sven,swordfish ChangeUser=sven,sesame</code></li> <li>▪ <code>csadm LogonPass=nils,ask ChangeUser=nils,ask</code></li> </ul>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---



When the credentials of a user are changed and the new credentials are equal to the old credentials, then the system returns an error and displays the message: "The specified user credentials are already in use by the specified user."

## Changing another user's HMAC password

If a user with user management rights changes the authentication credentials (using the `csadm ChangeUser` command) of another user who has HMAC password authentication, this other user cannot perform commands he/she needs an authentication for (except for the `csadm ChangeUser` command). The `csadm ListUser` command shows this user with the `I[1]` attribute value. To enable the user to perform commands he/she needs an authentication for, the user must change his/her credentials. To do so, this user must perform the `csadm ChangeUser` command. If the user is a PKCS#11 Security Officer (SO) or a PKCS#11 normal user, he/she may perform the `p11tool2 SetPIN` command as an alternative. It is important that the changes are performed by the user himself/herself and not, for example, by an administrator. The `csadm ListUser` command then shows this user with the `I[0]` attribute value.

Example procedure:

1. Show which users are available.

```
csadm Dev=194.167.1.3 ListUser
```

Example output:

Name	Permission	Mechanism	Attributes
ADMIN	22000000	RSA sign	Z[0]I[0]
test	22000000	HMAC passwd	Z[0]I[0]

2. Verify that the test user is capable of performing a command he/she needs an authentication for, for example, by performing a `csadm MBKListKeys` command.

```
csadm Dev=194.167.1.3 LogonPass=test,12345678 MBKListKeys
```

Example output:

slot	name	len	algo	type	k	generation	date	key	check	value
3	<NoName>	32	AES	XOR	2	2007/08/06	10:35:50	106B5E4E84031BDE		
7	AUTO-GEN	32	AES	SHARE	1	2023/08/07	10:11:11	10785E4E84032ADE		

3. Let the admin user change the test user's credentials.

```
csadm Dev=194.167.1.3 LogonSign=ADMIN,:cs2:cjo:USB0  
ChangeUser=test,11223344
```

4. The test user now is shown with the `I[1]` attribute, i.e., he/she cannot perform commands he/she needs an authentication for (except for the `csadm ChangeUser` command).

```
csadm Dev=194.167.1.3 ListUser
```

Example output:

Name	Permission	Mechanism	Attributes
ADMIN	22000000	RSA sign	Z[0]I[0]
test	22000000	HMAC passwd	Z[0]I[1]

5. This is why the test user cannot perform a command he/she needs an authentication for, for example, a `csadm MBKListKeys` command.

```
csadm Dev=194.167.1.3 LogonPass=test,11223344 MBKListKeys
```

Example output:

```
Error B0830091  
CryptoServer module CMDS, Command scheduler  
The user credentials need to be updated
```

6. The test user must change his/her credentials.

```
csadm Dev=194.167.1.3 LogonPass=test,11223344 ChangeUser=test,12345678
```

7. The test user is now shown with the `I[0]` attribute, i.e., he/she can perform commands he/she needs an authentication for.

```
csadm Dev=194.167.1.3 ListUser
```

Example output:

Name	Permission	Mechanism	Attributes
ADMIN	22000000	RSA sign	Z[0]I[0]
test	22000000	HMAC passwd	Z[0]I[0]

8. Verify that the test user can perform commands he/she needs an authentication for, for example, by performing a `csadm MBKListKeys` command.

```
csadm Dev=194.167.1.3 LogonPass=test,12345678 MBKListKeys
```

Example output:

slot	name	len	algo	type	k	generation	date	key	check	value
3	<NoName>	32	AES	XOR	2	2007/08/06	10:35:50	106B5E4E84031BDE		
7	AUTO-GEN	32	AES	SHARE	1	2023/08/07	10:11:11	10785E4E84032ADE		

#### 4.6.4 DeleteUser

This function deletes a user from the user database.

In order to prevent the device from getting non-administrable, the sum of the permissions of the remaining users, who use a signature-based authentication mechanism, has to be at least level 2 in user group 7 and level 1 in group 6. If this is not the case, the function will refuse execution and return an error code.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; DeleteUser=&lt;user&gt;</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 7 (20000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<user>	Existing user name

<b>Example</b>	<code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 DeleteUser=sven</code>
----------------	--

**Output**

Upon successful execution of the command, no output is given.

## 4.6.5 BackupUser

This command creates a file with a backup of all users that currently exist on the device. Each user entry in the created backup file is encrypted with the device Master Backup Key (MBK) and therefore can be handled without additional security measures.



Perform the `csadm MBKListKeys` command to determine which Master Backup Key (MBK) is currently in use in MBK slot 3 by the device. This MBK is used by the `csadm BackupUser` command to protect the backup file to be generated. If the MBK in MBK slot 3 is the autogenerated MBK named `AUTO-GEN`, the `csadm BackupUser` command cannot be performed. Import a different MBK into MBK slot 3 using the `csadm MBKImportKey` command.

It is important to note down which MBK has been used because for a successful restoring of this backup file at a later date it is necessary that the same MBK is in MBK slot 3. Otherwise, for example, after the execution of a `csadm MBKImportKey` command or after an MBK rollover, the backup file is inaccessible.



If the device version is < V4.30, the user ADMIN is not included in the backed up file

If user data is backed up using an old firmware not supporting the `I[]` attribute and this user data is restored using a new firmware supporting the `I[]` attribute, the restored users do not have the `I[]` attribute, i.e., the restored users do not need to change their credentials.

If users having the `I[]` attribute are backed up and restored, the restored users have the `I[]` attribute in the state they had before the backup.

You cannot downgrade the `I[]` attribute, i.e., perform a `csadm BackupUser` command for a user having the `I[]` attribute and then perform a `csadm RestoreUser` command on an old firmware not supporting the `I[]` attribute. This procedure would cause a `Bad user attribute` error (B0830017).

For more details about the `I[]` attribute, see [ListUser \(p. 114\)](#).

As of SecurityServer 6.0.0, a new user backup structure/format has been introduced.



- For creating and restoring backups, it is recommended to use a csadm version that matches the firmware version.
- Backups with the format earlier than 6.0.0 are still possible to be restored, except in FIPS mode because FIPS does not allow the old format.
- Restoring a backup with the new format into firmware version < 6.0.0 is not possible. Generally, restoring backups of a newer firmware version into older firmware is not supported.
- As of version 6.0.0, customers must use the `csadm BackupUser` command to back up users, because `csadm ... BackupDatabase=user.db` is not supported anymore.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; BackupUser=&lt;file&gt;[,&lt;flags&gt;]</code>
---------------	--

<b>Authentication</b>	The command must be authenticated with permission 2 in user group 7 (20000000)
-----------------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<file>	Path and file name of the backup file to be created
<flags>	<code>Overwrite</code> (Optional) Backup file is overwritten if already existing

<b>Example</b>	<code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 ... BackupUser=d:\temp\cs123456-20051212.ubk,overwrite</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---



If the 'overwrite' flag is not set and the backup file already exists, no backup is performed and the existing backup file is not overwritten.

### 4.6.6 RestoreUser

This command restores the users in the `user.db` database on the device from a backup file created by the `csadm BackupUser` command.



The same Master Backup Key (MBK) that has been used to create the backup file must be stored on the device.



To find out whether a user already exists in the `user.db` database on the device, perform the `csadm ListUser` command. Do not try to delete the user.

If user data is backed up using an old firmware not supporting the `I[]` attribute and this user data is restored using a new firmware supporting the `I[]` attribute, the restored users do not have the `I[]` attribute, i.e., the restored users do not need to change their credentials.

If users having the `I[]` attribute are backed up and restored, the restored users have the `I[]` attribute in the state they had before the backup.

You cannot downgrade the `I[]` attribute, i.e., perform a `csadm BackupUser` command for a user having the `I[]` attribute and then perform a `csadm RestoreUser` command on an old firmware not supporting the `I[]` attribute. This procedure would cause a `Bad user attribute` error (B0830017).

For more details about the `I[]` attribute, see *ListUser*.

As of SecurityServer 6.0.0, a new user backup structure/format has been introduced.

- For creating and restoring backups, it is recommended to use a csadm version that matches the firmware version.
- Backups with the format earlier than 6.0.0 are still possible to be restored, except in FIPS mode because FIPS does not allow the old format.
- Restoring a backup with the new format into firmware version < 6.0.0 is not possible. This procedure would cause an `Illegal length of command block` error (B0830008). Generally, restoring backups of a newer firmware version into older firmware is not supported.

- As of version 6.0.0, customers must use the `csadm BackupUser` command to back up users, because `csadm ... BackupDatabase=user.db` is not supported anymore.



As of u.trust Anchor FIPS 140-3 6.0.0, a new firmware and a new user data backup format are applied. As a consequence, if you have backed up user data using an u.trust Anchor < 6.0.0 (i.e. using the old firmware with the old user data backup format), this data backup cannot be restored in one step by u.trust Anchor FIPS 140-3 >= 6.0.0.

Instead, proceed as follows:

1. Restore the user data backup using u.trust Anchor non-FIPS >= 6.0.0.
2. Create a new user data backup using u.trust Anchor non-FIPS >= 6.0.0 (i.e. using the new firmware with the new user data backup format).
3. Restore the new user data backup using u.trust Anchor FIPS 140-3 >= 6.0.0 (i.e. using the new firmware with the new user data backup format).

If users cannot be restored, for example because they use a HASH that is no longer supported for HMAC authentication, they will be skipped during the restore.

Administrators get an output, how many users have been restored and listing those who could not be restored.

For example, skipping/not restoring invalid users prevents users who can no longer log in from ending up in the database.

#### Example output RestoreUser

```
Restored 5 out of 8 users in backup.
Skipped user user3 (B0890007)
Skipped user user7 (B0890007)
Skipped user Admin (User is currently logged in)
```

#### Syntax

```
csadm [Dev=<device>] <Authentication> RestoreUser=<file>[,<flags>]
```



#### Authentication

The command must be authenticated with permission 2 in user group 7 (20000000)

<b><i>Parameter</i></b>	<b><i>Description</i></b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<file>	Path and file name of the backup file to be created

Parameter	Description																
<flags>	The audit log entries <code>Delete User</code> and <code>Restore User</code> mentioned below are actually as follows: <code>FC:0x083 SFC:0x05 Delete User ‘&lt;user name&gt;’ [error code]</code> and <code>FC:0x083 SFC:0x0D Restore User ‘&lt;user name&gt;’ (&lt;short user name&gt;, &lt;authentication mechanism ID&gt; &lt;user’s permission&gt; &lt;access properties&gt;) [error code]</code> <code>&lt;access properties&gt;</code> contains a comma-separated list of strings of the form <code>&lt;name&gt;=&lt;value&gt;</code> . Examples: <code>CXI GROUP=SLOT_0001</code> <code>CXI_GROUP=test</code> <b>add (default)</b>																
	<table><tr><th>Scenario</th><th>Action</th><th>Audit Log Entry</th><th>Output</th></tr><tr><td>User in the backup file does not exist yet in the <code>user.db</code> database.</td><td>User is added</td><td><code>Restore User</code></td><td>User <code>&lt;username&gt;</code> added to <code>user.db</code></td></tr><tr><td>User in the backup file already exists with in the <code>user.db</code> database.</td><td>None</td><td>None</td><td>User <code>&lt;username&gt;</code> not added to <code>user.db</code> because the account already exists</td></tr><tr><td>User exists in the <code>user.db</code> database but not in the backup file.</td><td>None</td><td>None</td><td>None</td></tr></table>	Scenario	Action	Audit Log Entry	Output	User in the backup file does not exist yet in the <code>user.db</code> database.	User is added	<code>Restore User</code>	User <code>&lt;username&gt;</code> added to <code>user.db</code>	User in the backup file already exists with in the <code>user.db</code> database.	None	None	User <code>&lt;username&gt;</code> not added to <code>user.db</code> because the account already exists	User exists in the <code>user.db</code> database but not in the backup file.	None	None	None
	Scenario	Action	Audit Log Entry	Output													
	User in the backup file does not exist yet in the <code>user.db</code> database.	User is added	<code>Restore User</code>	User <code>&lt;username&gt;</code> added to <code>user.db</code>													
	User in the backup file already exists with in the <code>user.db</code> database.	None	None	User <code>&lt;username&gt;</code> not added to <code>user.db</code> because the account already exists													
	User exists in the <code>user.db</code> database but not in the backup file.	None	None	None													
	<b>overwrite</b>																
	<table><tr><th>Scenario</th><th>Action</th><th>Audit Log Entry</th><th>Output</th></tr><tr><td>User in the backup file does not exist yet in the <code>user.db</code> database.</td><td>User is added.</td><td><code>Restore User</code></td><td>User <code>&lt;username&gt;</code> added to <code>user.db</code></td></tr><tr><td>User in the backup file already exists with in the <code>user.db</code> database and user is not logged in.</td><td>User is overwritten.</td><td><code>Delete User</code> and <code>Restore User</code></td><td>User <code>&lt;user name&gt;</code> replaced in <code>user.db</code></td></tr><tr><td>User in the backup file already exists with in the <code>user.db</code> database and user is logged in.</td><td>None</td><td>None</td><td>User <code>&lt;user name&gt;</code> not replaced in <code>user.db</code> because the user is currently logged in</td></tr></table>	Scenario	Action	Audit Log Entry	Output	User in the backup file does not exist yet in the <code>user.db</code> database.	User is added.	<code>Restore User</code>	User <code>&lt;username&gt;</code> added to <code>user.db</code>	User in the backup file already exists with in the <code>user.db</code> database and user is not logged in.	User is overwritten.	<code>Delete User</code> and <code>Restore User</code>	User <code>&lt;user name&gt;</code> replaced in <code>user.db</code>	User in the backup file already exists with in the <code>user.db</code> database and user is logged in.	None	None	User <code>&lt;user name&gt;</code> not replaced in <code>user.db</code> because the user is currently logged in
	Scenario	Action	Audit Log Entry	Output													
	User in the backup file does not exist yet in the <code>user.db</code> database.	User is added.	<code>Restore User</code>	User <code>&lt;username&gt;</code> added to <code>user.db</code>													
User in the backup file already exists with in the <code>user.db</code> database and user is not logged in.	User is overwritten.	<code>Delete User</code> and <code>Restore User</code>	User <code>&lt;user name&gt;</code> replaced in <code>user.db</code>														
User in the backup file already exists with in the <code>user.db</code> database and user is logged in.	None	None	User <code>&lt;user name&gt;</code> not replaced in <code>user.db</code> because the user is currently logged in														

Parameter	Description			
replace	Scenario	Action	Audit Log Entry	Output
	User exists in the <code>user.db</code> database but not in the backup file.	None	None	None
	Scenario	Action	Audit Log Entry	Output
	User in the backup file does not exist yet in the <code>user.db</code> database.	User is added.	Restore User	User <user name> added to <code>user.db</code>
	User in the backup file already exists in the <code>user.db</code> database.	User is overwritten.	Delete User and Restore User	User <user name> replaced in <code>user.db</code>
	A single user manager is logged in and this user manager is included in the backup file or several user managers are logged in and at least one of these user managers is included in the backup file.	<code>error number != 0</code> and The restore operation is aborted and not a single user account is restored.	None	RestoreUser in 'replace' mode is not supported when logged-in user manager(s) shall be restored.
User exists in the <code>user.db</code> database but not in the backup file.	User is deleted. If this user is a single logged-in user manager, deleting this user manager is the last operation of the procedure, because deleting this user manager terminates the Secure Messaging session.	Delete User	User <user name> removed from HSM	

Parameter	Description
	<div>  <p>If the deleted user is a single logged-in user manager, deleting this user manager is the last operation of the procedure, because deleting him terminates the Secure Messaging session. If you have a new device, the ADMIN user is the only existing user. If the backup file does not include the ADMIN user and you perform the <code>csadm RestoreUser</code> command, all users from the backup file are added to the <code>user.db</code> database and then the ADMIN user is deleted.</p> </div> <div>  <p>If a single user manager is logged in and this user manager is included in the backup file, the <code>csadm RestoreUser</code> command returns an error (error number <math>\neq 0</math>). The restore operation is aborted and not a single user account is restored. The same applies if several user managers are logged in and at least one of these user managers is included in the backup file.</p> </div>
<b>Example</b>	<code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 RestoreUser=d:\temp\cs123456-20051212.ubk,overwrite</code>
<b>Output</b>	Upon successful execution of the command, an indication of the executed action is returned as stated in the flag table above.

### 4.6.7 SetMaxAuthFails

This command defines the maximum number (n) of consecutive failed authentication attempts for each user. Optionally, the maximum can be set for users with HMAC password authentication only.

If this maximum number is reached the user is blocked. A blocked user is not able to authenticate towards the device anymore.

A blocked user can be unblocked by a user with user management rights (minimum permission 2 in the user group 7; 20000000) who is logged on to the device.

`SetMaxAuthFails` is a global configuration, i.e., it applies to all users except if the `pwdOnly` parameter is set.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; SetMaxAuthFails=&lt;n&gt;[,pwdOnly]</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 7 (20000000)
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<n>	The maximum number of consecutive failed authentication attempts for each user (0 - 255.) A value of 0 means that an unlimited number of consecutive failed authentication attempts is allowed for each user. The default for n is 0.
<pwdOnly>	<code>pwdOnly</code> is available only for csadm version 2.5.1 or higher. If <code>pwdOnly</code> is set, the maximum number of consecutive failed authentication attempts is only applied to all users using HMAC password authentication. If <code>pwdOnly</code> is set and a user uses RSA or ECDSA signature authentication or RSA smartcard authentication, an unlimited number of consecutive failed authentication attempts is allowed for this user. The setting <code>pwdOnly</code> is equivalent to <code>pwdOnly=1</code> , <code>pwdOnly=y</code> , <code>pwdOnly=yes</code> and <code>pwdOnly=0</code> .

<b>Example</b>	<pre>csadm Dev=196.163.1.2 LogonSign=ADMIN,:cs2:cjo:USB0 SetMaxAuthFails=3 csadm Dev=196.163.1.2 LogonSign=ADMIN,:cs2:cjo:USB0 SetMaxAuthFails=3,pwdOnly csadm LogonPass=Sven,ask SetMaxAuthFails=4 csadm LogonPass=Sven,ask SetMaxAuthFails=4,pwdOnly</pre>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

#### 4.6.8 GetMaxAuthFails

This command displays the maximum number (n) of consecutive failed authentication attempts for each user. This number is a global configuration, i.e., it applies to all users.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] GetMaxAuthFails</code>
---------------	---



<b>Authentication</b>	The command must be authenticated with permission 2 in user group 7 (20000000)
-----------------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	csadm Dev=192.168.1.2 GetMaxAuthFails
----------------	---------------------------------------

<b>Output</b>	<p>Upon successful execution of the command, the following information is returned:</p> <ul style="list-style-type: none"> <li>▪ <code>n (0...255)</code> The maximum number (n) of consecutive failed authentication attempts for each user. If this number is reached the user is blocked and unable to authenticate towards the device anymore. A value of 0 means that an unlimited number of consecutive failed authentication attempts is allowed for each user. The default for n is 0.</li> <li>▪ <code>n, password users only</code> This output is available only for csadm version 2.5.1 or higher. The maximum number (n) of consecutive failed authentication attempts is only applied to users using HMAC password authentication. If a user uses RSA or ECDSA signature authentication or RSA smartcard authentication, an unlimited number of consecutive failed authentication attempts is allowed for this user.</li> </ul>
---------------	---

## 4.7 Commands for Managing the Master Backup Keys

To provide backup functionality, the device is able to store up to 256 Master Backup Keys (MBK slot 0...255; SecurityServer/CryptoServer SDK 4.20 or earlier: 4 Master Backup Keys corresponding to MBK slot 0...3) to be used by various applications. Each Master Backup Key (MBK) can either be a DES key (DES keys are supported for csadm versions < 2.5.3 only; 16 or 24 bytes) or an AES key (16, 24 or 32 bytes).

An MBK can be generated on the device and — split into key parts (key shares) — externally stored on two or more smartcards. The key parts of a MBK can be imported into the device either from smartcards or they can be manually keyed in at the PIN pad. If a new key should be imported into an MBK slot which already contains an MBK, the old MBK is overwritten with a new MBK without any verification.

In addition to the local management of Master Backup Keys, where the PIN pad has to be directly connected to the device, the MBK can also be managed remotely, without having direct physical access to the device (which perhaps is located in a data center). To protect the key parts during transmission from/to the device, they are encrypted with the session key

(AES) that was exchanged on establishment of the secure messaging session with the device. A secure messaging session is therefore necessary for remote MBK management.

This chapter describes how Master Backup Keys (MBK) can be managed remotely.



If you are using an external database, make sure that you do not use the automatically generated MBK. This MBK is shown in the output of the `csadm MBKListKeys` command in the `name` column as `AUTO-GEN`. If an alarm or an external erase occurs, this MBK is deleted and the external database is inaccessible.

Use an MBK instead that you have generated, backed up in keyfiles or on smartcards (`csadm MBKGenerateKey` command) and imported into the device (`csadm MBKImportKey` command). If now an alarm or an external erase occurs, reimport the same MBK from the keyfiles or smartcards.

If you access one external database from several devices, make sure that you use the same MBK in all these devices.

If you want to change the MBK an external database is encrypted with, perform an MBK rollover.



To unify the MBK storage all applications of Utimaco IS GmbH (e.g. CXI) use the following convention:

MBK slot 0	DES key (DES keys are supported for csadm versions < 2.5.3 only.)
MBK slot 1	Intermediate key (for example, while key is copied)
MBK slot 2	not used
MBK slot 3	AES key

We recommend using MBK slot 3 exclusively for any MBK storage (AES key). If the MBK shall be used to backup keys from firmware module CXI, then the MBK in MBK slot 3 is mandatory: an MBK from any other MBK slot is not accepted by CXI.

The MBK slot number must not be confused with the PKCS#11 slot number.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a `csadm` command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary

## 4.7.1 MBKListKeys

This command lists all MBKs currently stored on the device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; MBKListKeys</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 7 (20000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=194.167.1.3 LogonSign=ADMIN,:cs2:cjo:USB0 MBKListKeys</code>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, a list of all MBKs is returned.</p> <pre> slot name          len algo  type          k  generation date          key check value ----- 3      &lt;NoName&gt;    32  AES   XOR           2      07/08/06 10:35:50 106B5E4E84031BDE </pre>
---------------	---

For each MBK the following information is given:

Parameter	Description
slot	Gives the MBK slot number (0...255; SecurityServer/CryptoServer SDK 4.20 or earlier: 0...3). The MBK slot number must not be confused with the PKCS#11 slot number.
name	Gives the key name (or <NoName> if no key name is given). AUTO-GEN indicates an autogenerated MBK.
len	Gives the MBK key length in byte
algo	Gives the key algorithm
type	Indicates if the MBK was exported in two XOR halves (XOR) or in more than two key shares (SHARE)
k	Gives the number of shares that is needed to recombine the MBK (with k=2 in case of export in two XOR halves)
generation date	This is the date and time of generation of the MBK (up to seconds)

<b>Parameter</b>	<b>Description</b>
key check value	This is a key check value (ISO-hash MDC2 over MBK) which can be used to identify the MBK

Table 18: Meaning of the output parameters of csadm MBKListKeys



If an older firmware package is used (firmware module MBK version later than 2.2.1.0) the values of `k`, `generation date` and `key check value` are not shown.

## 4.7.2 MBKGenerateKey

This command generates an MBK on the device, but it does not store the MBK on the device, it splits the MBK into `n` key shares and transmits the encrypted key shares to the host. The encryption for the transmission is done with the session key. The key shares are decrypted on the host and stored either on `n` smartcards or in `n` keyfiles.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; Key=&lt;keyspec&gt; ... MBKGenerateKey=&lt;keytype&gt;,&lt;keylen&gt;[,&lt;n&gt;,&lt;k&gt;,&lt;keyname&gt; ]</code>
---------------	---

<b>Authentication</b>	Permission level 2 in the user group 6 (02000000) Additionally, it is mandatory to execute the command within a secure messaging session. This is automatically fulfilled if the authentication command <code>LogonSign</code> or <code>LogonPass</code> is used.
-----------------------	--

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

Parameter	Description
<keyspec>	<p>Key specifier where the generated MBK shares should be stored</p> <ul style="list-style-type: none"> <li>▪ <code>&lt;smartcard,record number&gt;</code>            Example: <code>cs2:cjo:USB0,15</code>            The record number specifies the location where to store the MBK share on the smartcard. Value range: 1...16. As a consequence, you can store up to 16 MBK shares on a smartcard. The default record number for AES keys is 15. (For csadm version earlier than 2.5.3, DES keys are supported as well. The default record number for DES keys is 1.) An MBK share stored on a smartcard cannot be deleted. It can only be overwritten.            If another MBK share is already stored in the specified record, it is overwritten by the MBK share to be generated. All MBK shares are stored in records with the same record number on different smartcards. Verify whether these records are free or can be overwritten by performing the <code>csadm MBKCardInfo</code> command for all used smartcards.</li> <li>▪ List of filenames and (if the key should be password protected) passwords-  <code>&lt;file#password&gt;</code> (for example, <code>file1#pwd1,file2#pwd2</code> ).               <ul style="list-style-type: none"> <li>• If the password is given as the string ask, hidden password entry is performed.</li> </ul> </li> </ul>
<keytype>	Key type of MBK: either DES (csadm versions < 2.5.3 only) or AES
<keylen>	Key length of MBK: 16, 24 (DES (csadm versions < 2.5.3 only), AES) or 32 bytes (AES only)
<n>	<ul style="list-style-type: none"> <li>▪ Number of key shares to be generated (default: 2):            If <code>&lt;n&gt;</code> is equal to 2, the key type is XOR (XOR half). If it is greater than 2, the key type is SHARE (key share).</li> </ul>
<k>	Number of key shares required to recombine key (default: 2)
<keyname>	<p>Key name (up to 8 characters with ANSI / ISO-8859-1 encoding).            Do not name the MBK to be created <code>AUTO-GEN</code> because this is the name of the autogenerated MBK.</p>

<b>Example</b>	<ul style="list-style-type: none"> <li>▪ <code>csadm versions &lt; 2.5.3 only:csadm LogonSign=ADMIN,:cs2:cjo:USB0 Key=:cs2:cjo:USB0,1 MBKGenerateKey=DES,24,2,2,MyDESKey</code></li> <li>▪ <code>csadm LogonPass=sven,mypassword Key=mbk1.key#swordfish,mbk2.key#sesame MBKGenerateKey=AES,32,2,2,MyAESKey</code></li> <li>▪ <code>csadm LogonPass=sven,ask Key=:cs2:cjo:USB0,15 MBKGenerateKey=AES,32,5,3,AES_new</code></li> </ul>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

- If smartcards are used (recommended):  
The PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.  
Watch the display of the PIN pad for instructions on further command processing.
- If keyfiles are used:  
We strongly recommend using encrypted keyfiles.



The newly generated MBK is not stored on the device. To store the MBK on the device it has to be imported with the command `MBKImportKey`.



To protect the key parts during transmission from/to the device, they are encrypted with the session key (AES) that was exchanged on establishment of the secure messaging session with the device.

A secure messaging session is therefore necessary for remote MBK management.

### 4.7.3 MBKImportKey

This command imports the key shares of an MBK either from two or more smartcards or from two or more keyfiles.



If you use SecurityServer/CryptoServer SDK 4.10, and you import a new MBK, the external key and key backups become inaccessible. The error message “invalid mac of key blob” (error code: 0xB0680026) is created.

This applies as well if you have upgraded to SecurityServer/CryptoServer SDK 4.20 or later after you have imported the original MBK.

If you use SecurityServer/CryptoServer SDK 4.20 and you import a new MBK, the external key and key backups become inaccessible. The error message “key blob encrypted with different MBK” (error code: 0xB0680080) is created.



Before you perform the `csadm MBKImportKey` command, verify which backup files have been generated because they might become inaccessible after the import. After having performed the `csadm MBKImportKey` command, regenerate these backup files.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; Key=&lt;keyspec&gt; MBKImportKey=&lt;slot_no&gt;</code>
---------------	---

<b>Authentication</b>	Permission level 2 in the user group 6 (02000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<keyspec>	<p>Key specifier, where the new MBK should be loaded from</p> <ul style="list-style-type: none"> <li>▪ <code>&lt;smartcard,record number&gt;</code> Example: <code>cs2:cjo:USB0,15</code> The record number specifies the location where to store the MBK share on the smartcard. Value range: 1...16. As a consequence, you can store up to 16 MBK shares on a smartcard. The default record number for AES keys is 15. (For csadm version earlier than 2.5.3, DES keys are supported as well. The default record number for DES keys is 1.) An MBK share stored on a smartcard cannot be deleted. It can only be overwritten. If another MBK share is already stored in the specified record, it is overwritten by the MBK share to be generated.</li> <li>▪ list of filenames and (if the key should be password protected) <code>passwords-&lt;file#password&gt;</code> (for example, <code>file1#pwd1,file2#pwd2</code>). <ul style="list-style-type: none"> <li>• If password is given as the string ask, hidden password entry is performed</li> <li>• If no password is given, the keyfile is stored in plaintext</li> </ul> </li> </ul>

Parameter	Description
<slot_no>	<p>MBK slot number on the device the MBK should be imported to; must be 3 for AES key (recommended; even mandatory for usage with firmware module CXI) and 0 for DES key (DES keys are supported for csadm versions &lt; 2.5.3 only). Supported value range: 0...255; SecurityServer/CryptoServer SDK 4.20 or earlier: 0...3.</p> <p>If there is already an old MBK in the MBK slot indicated by &lt;slot_no&gt;, this old MBK is overwritten by the new MBK to be imported. The only exception is the case that &lt;slot_no&gt; is 3 and the old MBK is an autogenerated MBK (shown as the MBK name AUTO-GEN in the output of the csadm MBKListKeys command). In this case, the autogenerated MBK is moved from MBK slot 3 to MBK slot 7 (or to the next free MBK slot &gt; 7 if MBK slot 7 is already occupied by another MBK) and the new MBK is imported into MBK slot 3.</p> <p>To verify which MBK is in a certain MBK slot, use the csadm MBKListKeys command.</p> <p>Consider that it is important which MBK is in use in MBK slot 3 by the device because this MBK is used by the csadm BackupUser command and the csadm BackupDatabase command to protect the backup files that are generated by these commands.</p> <p>Consider that it is important which MBK is in use in MBK slot 3 by the device because this MBK is used by the csadm BackupDatabase command to protect the backup files that are generated by these commands.</p> <p>It is important to note down which MBK has been used for these backup commands because for a successful restoring of these backup files at a later date it is necessary that the same MBK is in MBK slot 3. Otherwise, for example, after the execution of a csadm MBKImportKey command or after an MBK rollover, the backup files are inaccessible.</p> <p>The MBK slot number must not be confused with the PKCS#11 slot number.</p>

Example	<pre>csadm LogonSign=ADMIN,:cs2:cjo:USB0 Key=:cs2:cjo:USB0,1 MBKImportKey=3 csadm LogonPass=sven,mypassword Key=mbk1.key#swordfish,mbk2.key#sesame MBKImportKey=3 csadm LogonPass=sven,ask Key=:cs2:cjo:USB0,15 MBKImportKey=3</pre> <p>Authenticate the csadm MBKImportKey command by the Admin1 user and the Admin3 user and import the MBK whose key shares are in the record number 2 of the smartcard containing the first MBK share and the record number 1 of the smartcard containing the second MBK share.</p> <pre>csadm Dev=PCI:0 LogonSign=Admin1,:cs2:cjo:USB0 LogonSign=Admin3,:cs2:cjo:USB0 Key=:cs2:cjo:USB0,2,:cs2:cjo:USB0,1 MBKImportKey=3</pre>
---------	---

Output	Upon successful execution of the command, no output is given.
--------	---

- If smartcards are used:  
A PIN pad can be connected to the computer, where the csadm tool is running (USB



port) or to another computer-

Watch the display of the PIN pad for instructions on further command processing.

- If keyfiles are used:

We strongly recommend using encrypted keyfiles and a hidden password entry.



To protect the key parts during transmission from/to the device, they are encrypted with the session key (AES) that was exchanged on establishment of the secure messaging session with the device.


A secure messaging session is therefore necessary for remote MBK management.

#### 4.7.4 MBKCopyKey

This command copies one key share of a MBK from one storage location to another.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

Syntax
<code>csadm Key=&lt;keyspec_source&gt; MBKCopyKey=&lt;keyspec_target&gt;</code>

<b>Parameters</b>	<ul style="list-style-type: none"> <li>▪ <code>&lt;keyspec_source &gt;</code> Key specifier, where the key share should be loaded from</li> <li>▪ <code>&lt;keyspec&gt;</code> Key specifier where the generated MBK shares should be stored <ul style="list-style-type: none"> <li>• <code>&lt;smartcard,record number&gt;</code> Example: <code>cs2:cjo:USB0,15</code> The record number specifies the location where to store the MBK share on the smartcard. Value range: 1...16. As a consequence, you can store up to 16 MBK shares on a smartcard. The default record number for AES keys is 15. (For csadm version earlier than 2.5.3, DES keys are supported as well. The default record number for DES keys is 1.) An MBK share stored on a smartcard cannot be deleted. It can only be overwritten.</li> </ul> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;">  If another MBK share is already stored in the specified record, it is overwritten by the MBK share to be generated. </div> <ul style="list-style-type: none"> <li>• List of filenames and (if the key should be password protected) <code>passwords- &lt;file#password&gt;</code> (for example, <code>file1#pwd1, file2#pwd2</code>). <ul style="list-style-type: none"> <li>• If password is given as the string <code>ask</code>, hidden password entry is performed.</li> <li>• If no password is given, the keyfile is stored in plaintext.</li> </ul> </li> <li>▪ <code>&lt;keyspec_target &gt;</code> Key specifier, where key share should be stored</li> </ul> </li></ul>
<b>Authentication</b>	None
<b>Examples</b>	<pre>csadm Key= mbk1.key#swordfish MBKCopyKey=:cs2:cjo:USB0,15 csadm Key=:cs2:cjo:USB0,15 MBKCopyKey=mbk2.key#ask</pre>
<b>Output</b>	None on success or error message

With this command the storage location of an MBK can be migrated from a smartcard to a keyfile or vice versa.

- If smartcards are used:  
A PIN pad (smartcard reader with keyboard and display) can be connected to the computer, where the csadm tool is running (USB port) or to another computer. Watch the display of the PIN pad for instructions on further command processing.

- If keyfiles are used:  
We strongly recommend using encrypted keyfiles.  
If encrypted keyfiles are used, we strongly recommend using a hidden password entry

### 4.7.5 MBKCardInfo

This command lists the contents of an MBK smartcard. Here, every MBK key share is stored in a separate record. The PIN pad can be connected to the host computer (USB port), where the csadm tool is running or to another computer.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

If you want to retrieve information not only about MBK shares but also about RSA key shares, ECDSA key shares, RSA keys and ECDSA keys stored on the smartcard, perform the `csadm GetCardInfo` command instead of performing the `csadm MBKCardInfo` command.

<b>Syntax</b>	<code>csadm MBKCardInfo=&lt;cardspec&gt;</code>
---------------	---

Parameter	Description
<cardspec>	Smartcard specifier of the MBK smartcard

<b>Example</b>	<code>csadm MBKCardInfo=:cs2:cjo:USB0</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, the card information is returned. An MBK smartcard may contain up to 16 records/MBK parts. For each stored key share the following information is given:										
	REC	TYPE	ALG	LEN	NAME	TIMEGEN		K	ID	HASH	
	-----										
	--										
	03	SHARE	AES	256	Test	13.04.2007 16:17:01		3	2	397620CEB7C61DBE	
	13	SHARE	AES	256	MbkAes28	09.05.2019 09:10:15		2	1	37BAFABF49C90D0C	
	14	XOR	AES	256	MbkAes29	09.05.2019 09:04:11		0	0	2722BDA0D7D6E821	
	15	XOR	AES	256	MbkAes30	09.05.2019 08:57:24		0	0	D0779FB705960D8A	

Parameter	Description
REC	Record number

<b>Parameter</b>	<b>Description</b>
TYPE	Share type <b>XORB</b> : XOR half <b>SHARE</b> : Key share
ALG	MBK algorithm
LEN	Key length in bits
NAME	Name of an MBK or <code>&lt;NoName&gt;</code> if no key name is given
TIMEGEN	Date and time of key generation
M	Number of shares that are necessary to recombine MBK Only relevant if <b>TYPE</b> is <b>SHARE</b>
ID	ID of the share
HASH	Check value over MBK (usually: first 8 bytes of ISO-hash MDC2 over MBK; if <b>ID</b> = <b>0</b> or <b>1</b> : first respectively second 8 bytes of ISO-hash MDC2 over MBK).

Table 19: Meaning of the output parameters of csadm MBKCardInfo

### 4.7.6 MBKCardCopy

This command copies the whole content of a MBK smartcard to another.

If records on the target smartcard already exist they will be overwritten without additional query. This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

<b>Syntax</b>	<code>csadm MBKCardCopy=&lt;keyspec&gt;</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;keyspec&gt;</code>	Key specifier of the smartcard

<b>Example</b>	<code>csadm MBKCardCopy=:cs2:cjo:USB0</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

### 4.7.7 MBKPINChange

This command changes the PIN on a smartcard where an MBK is stored.

This command is executed locally without a connection to a device. Therefore, the state or mode of any underlying device is irrelevant for the command execution, and no authentication at any device is necessary.

<b>Syntax</b>	<code>csadm MBKPINChange=&lt;cardspec&gt;</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;cardspec&gt;</code>	Smartcard specifier of the MBK smartcard

<b>Example</b>	<code>csadm MBKPINChange=:cs2:cjo:USB0</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---



The PIN pad can be connected to the computer where the csadm tool is running (USB port) or to another computer.

Please follow the instructions on the display of the PIN pad.

## 4.8 Commands for Firmware Management

This chapter describes the commands to be used for managing the firmware on the device, i.e., for showing the list of firmware modules and their versions currently loaded on the device, loading and deleting firmware modules, updating the firmware, etc.

For performing these commands, the device has to be either in *Operational Mode* or in *Maintenance Mode*.

### 4.8.1 ListFirmware

This function returns a list with information about all firmware modules that are currently running, giving the firmware module ID, abbreviated name, CPU target `type` the firmware module is compiled for ( `C64+` for CSe-Series, `C64` for Se-Series Gen2, `SDK` for Simulator for

Windows and SIM for Simulator for Linux), version number and the respective firmware module `initialization level`.



If a module cannot be started or fully initialized, the `csadm GetBootLog` command provides more detailed information about the reason.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] ListFirmware</code>
<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>Example</b>	<code>csadm Dev=192.168.1.1 ListFirmware</code>

Upon successful execution of the command, a list of all firmware modules is returned. The example output may differ from the output on your CryptoServer with respect to the listed firmware modules and/or their versions depending on the installed firmware package and its version.

If for a firmware module no entry is found, the module is not loaded.



After loading or replacing a firmware module, the device has to be restarted with the `Restart` command before the firmware module becomes active.

For CryptoServer 4.45 and later, the following applies:

The implementation of cryptographic primitives has been optimized for, and harmonized across, Utimaco's HSM platforms and host components, resulting in a general performance increase of cryptographic operations. The exact improvements depend on algorithm, key size, size of data to be encrypted/signed, and HSM model. The improvements are highest for short-running operations like AES encryption of small blocks of data. This new common cryptographic library is delivered as CRYPT firmware module. In a firmware package, this CRYPT firmware module replaces the following modules that were previously handled as separate modules: AES, HASH, DSA, VRSA, LNA, ECDSA, ECA, POST and ASN.1. The modules AES, HASH, etc. appear as active firmware modules in addition to the new CRYPT module when calling the `csadm ListFirmware` command.

Example output for CryptoServer 4.45 and later:

ID	name	type	version	initialization level
0	SMOS	SDK	5.6.4.1	INIT_OK
4	POST	SDK	2.2.1.0	INIT_OK
68	CXI	SDK	2.4.11.1	INIT_OK
81	VDES	SDK	2.2.1.0	INIT_OK
82	PP	SDK	1.4.1.2	INIT_OK
83	CMDS	SDK	3.8.4.1	INIT_OK
84	VRSA	SDK	2.2.1.0	INIT_OK
85	SC	SDK	1.2.0.7	INIT_OK
86	UTIL	SDK	3.0.7.1	INIT_OK
87	ADM	SDK	3.1.2.0	INIT_OK
88	DB	SDK	2.0.0.2	INIT_OK
89	HASH	SDK	2.2.1.0	INIT_OK
8a	STUN	SDK	0.0.0.1	INIT_OK
8b	AES	SDK	2.2.1.0	INIT_OK
8d	DSA	SDK	2.2.1.0	INIT_OK
8e	LNA	SDK	2.2.1.0	INIT_OK
8f	ECA	SDK	2.2.1.0	INIT_OK
91	ASN1	SDK	2.2.1.0	INIT_OK
96	MBK	SDK	2.5.1.1	INIT_OK
9a	NTP	SDK	1.2.1.1	INIT_OK
9c	ECDSA	SDK	2.2.1.0	INIT_OK
9f	CRYPT	SDK	2.2.1.0	INIT_OK

Example output for CryptoServer earlier than 4.45:

### Output

ID	name	type	version	initialization level
0	SMOS	C64	3.1.1.2	INIT_OK
a	HCE	C64	1.0.1.0	INIT_OK
e	BCM	C64	1.0.2.0	INIT_OK
64	PKCS11	C64	1.1.3.0	INIT_OK
68	CXI	C64	2.1.0.1	INIT_OK
81	VDES	C64	1.0.4.0	INIT_OK
82	PP	C64	1.2.3.6	INIT_OK
83	CMDS	C64	3.0.3.5	INIT_OK
84	VRSA	C64	1.1.1.2	INIT_OK
85	SC	C64	1.2.0.0	INIT_OK
86	UTIL	C64	3.0.1.2	INIT_OK
87	ADM	C64	3.0.7.1	INIT_OK
88	DB	C64	1.1.2.4	INIT_OK
89	HASH	C64	1.0.7.0	INIT_OK
8b	AES	C64	1.0.5.0	INIT_OK
8d	DSA	C64	1.0.0.0	INIT_OK
8e	LNA	C64	1.1.0.0	INIT_OK
8f	ECA	C64	1.1.2.0	INIT_OK
91	ASN1	C64	1.0.3.3	INIT_OK
96	MBK	C64	2.2.3.2	INIT_OK
9a	NTP	C64	1.2.0.6	INIT_OK
9c	ECDSA	C64	1.1.1.1	INIT_OK

The meaning of the initialization levels is the following:

<b>Initiali- zation level</b>	<b>Description</b>
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 memory (for CSe-Series: DDR2 RAM memory).
INIT_ INTER- NAL	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.
INIT_ DEP_ O K	The module has successfully completed the check of dependencies on other modules (second step of initialization). "Dependencies on other modules" means that the module depends on services provided by other modules in order to run correctly. Example: the SC (Smartcard) module requires the PP (PIN pad) module to do its work.
INIT_ OK	This is the highest possible level: The initialization of the module is successfully completed (third step of initialization). Possible calls to services from other modules are done successfully.
INIT_ FAILE D	Module initialization failed. Services provided by this module are not available.



<b>Initiali zation level</b>	<b>Description</b>
INIT_ INACT IVE	Module is loaded but cannot supply services, for example, because of not-mounted hardware components. This initialization level is currently only used by the firmware modules HCE and EXAR (relevant for Se-Series Gen2 only).
SUSPE NDED	Module was shut down and cannot supply services anymore.

Table 20: Initialization levels

## 4.8.2 ListFiles

This command lists all files stored on the device. The following directories are available on the different storage devices:

<b>Device</b>	<b>Directory</b>	<b>Size</b>	<b>Description</b>
FLASH	\FLASH	32 MByte	Every firmware module has read and write access to this working directory. The regular set of firmware modules and any other kind of application data (databases, configuration or log files) is stored here.
NVRAM (non volatile RAM)	\NVRAM	512 kByte	Every firmware module has read and write access to this directory. Frequently changing data, which shall be stored non-volatile, should be stored here.

Table 21: Directories on the device



Do not use the `\FLASH` directory to store often changing data (for example, counter). This will result in a great wear and tear of the flash-component and therefore in a decreased lifetime of the device.

Use the `\NVRAM` directory instead to store this data non-volatile.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] ListFiles[=&lt;mode&gt;]</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Parameter</b>	<b>Description</b>
<mode>	<ul style="list-style-type: none"><li>▪ &lt;none &gt; - Show all non-hidden files.</li><li>▪ FLASH - Only non-hidden files in the \FLASH directory are listed.</li><li>▪ SYS - Only hidden files in the \FLASH directory are listed.</li><li>▪ NVRAM - Only files in the \NVRAM directory are listed.</li></ul>

<b>Example</b>	<ul style="list-style-type: none"><li>▪ csadm ListFiles</li><li>▪ csadm ListFiles=FLASH</li><li>▪ csadm ListFiles=NVRAM</li><li>▪ csadm ListFiles=SYS</li></ul>
----------------	---

**Output**

Upon successful execution of the command, a list of all files is returned.  
 The example output may differ from the output on your CryptoServer with respect to the listed firmware modules, their sizes and/or versions depending on the installed firmware package and its version.  
 For CryptoServer 4.45 and later, the following applies:  
 The implementation of cryptographic primitives has been optimized for, and harmonized across, Utimaco's HSM platforms and host components, resulting in a general performance increase of cryptographic operations. The exact improvements depend on algorithm, key size, size of data to be encrypted/signed, and HSM model. The improvements are highest for short-running operations like AES encryption of small blocks of data. This new common cryptographic library is delivered as CRYPT firmware module. In a firmware package, this CRYPT firmware module replaces the following modules that were previously handled as separate modules: AES, HASH, DSA, VRSA, LNA, ECDSA, ECA, POST and ASN.1. The output of a `csadm ListFiles` command therefore shows the CRYPT module instead of separate AES, HASH, etc. modules.  
 Example output for CryptoServer 4.45 and later:

file name	size	module:	ID	type
version	name			
-----				
-----				
FLASH\CXIKEY.db	20	-		
FLASH\VMBK1.db	255	-		
FLASH\adm.msc	45740	ADM	0x087	SDK
3.1.2.0	Administration Module	SDK		
FLASH\audit000000.log	2053	-		
FLASH\cmds.msc	57516	CMDS	0x083	SDK
3.8.4.1	Command Scheduler	(SDK)		
FLASH\crypt.msc	304300	CRYPT	0x09f	SDK
2.2.1.0	CRYPT Module	SDK		
FLASH\cxi.msc	182956	CXI	0x068	SDK
2.4.11.1	Crypto eXtended Interf.	SDK		
FLASH\db.msc	23212	DB	0x088	SDK
2.0.0.2	Database module	(SDK)		
FLASH\mbk.msc	38060	MBK	0x096	SDK
2.5.1.1	Master Backup Key Module	SDK		
FLASH\ntp.msc	13996	NTP	0x09a	SDK
1.2.1.1	TimeAdjust Module	SDK		
FLASH\pp.msc	35500	PP	0x082	SDK
1.4.1.2	PIN pad Driver	(SDK)		
FLASH\sc.msc	20140	SC	0x085	SDK
1.2.0.7	Smartcard module	(SDK)		
FLASH\smos.msc	117420	SMOS	0x000	SDK
5.6.4.1	Operating System	SDK		
FLASH\stun.msc	20652	STUN	0x08a	SDK
0.0.0.1	STUN Module	SDK		
FLASH\user.db	343	-		
FLASH\util.msc	14508	UTIL	0x086	SDK
3.0.7.1	Utility Module	SDK		
16 files 876671 bytes, 65421824 bytes available				

Example output for CryptoServer earlier than 4.45:

file name	size	module:
ID type version name		
-----		
FLASH\CXIKEY.db	64646 -	
FLASH\VMBK1.db	97 -	
FLASH\adm.msc	43692 ADM	0x087 SDK
3.0.32.0	Administration Module SDK	
FLASH\aes.msc	43692 AES	0x08b SDK
1.4.2.0	AES Module SDK	
FLASH\asn1.msc	15532 ASN1	0x091 SDK
1.0.3.6	Asn1 Module SDK	
FLASH\audit000000.log	2190 -	
FLASH\auditkey.db	1785 -	
FLASH\authkey.db	1785 -	
FLASH\cmds.msc	53932 CMDS	0x083 SDK
3.6.2.0	Command Scheduler (SDK)	
FLASH\cxi.msc	172204 CXI	0x068 SDK
2.3.1.0	Crypto eXtended Interf.SDK	
FLASH\db.msc	23212 DB	0x088 SDK
1.3.2.4	Database module (SDK)	
FLASH\dsa.msc	28332 DSA	0x08d SDK
1.2.3.3	DSA Module SDK	
FLASH\eca.msc	67756 ECA	0x08f SDK
1.1.14.0	Elliptic Curve Arith. (SDK)	
FLASH\ecdsa.msc	42156 ECDSA	0x09c SDK
1.1.21.0	ECDSA Module SDK	
FLASH\exmp.db	20 -	
FLASH\exmp.msc	19116 EXMP	0x100 SDK
2.1.0.0	Example Module SDK	
FLASH\hash.msc	38060 HASH	0x089 SDK
1.0.12.3	Hash Module SDK	
FLASH\lna.msc	39084 LNA	0x08e SDK
1.2.4.7	Long Number Arithmetic (SDK)	
FLASH\mbk.msc	34476 MBK	0x096 SDK
2.3.0.0	Master Backup Key Module SDK	
FLASH\mdlsigalt.key	267 -B9CB966B7A-	
FLASH\post.msc	32428 POST	0x004 SDK
1.0.0.2	POST Module SDK	
FLASH\pp.msc	34476 PP	0x082 SDK
1.3.2.0	PIN pad Driver (SDK)	
FLASH\sc.msc	21676 SC	0x085 SDK
1.2.0.4	Smartcard module (SDK)	
FLASH\smos.msc	113836 SMOS	0x000 SDK
5.6.1.0	Operating System SDK	
FLASH\swap.com	0 -	
FLASH\user.db	4752 -	
FLASH\util.msc	13996 UTIL	0x086 SDK
3.0.5.3	Utility Module SDK	
FLASH\vdes.msc	23212 VDES	0x081 SDK
1.0.10.0	DES Module SDK	
FLASH\vdv.msc	15532 VDX	0x0f5 SDK
1.0.0.2	Vendor Defined Example	
FLASH\vrta.msc	47276 VRSA	0x084 SDK
1.3.6.7	RSA Module SDK	

<i>Parameter</i>	<i>Description</i>
file name	Path\name of the file
size	Size of the file
<b>Additional parameters for firmware modules</b>	
module	Abbreviation of the name of the firmware module)
ID (FC)	Firmware module ID (function code)
type	<ul style="list-style-type: none"> <li>▪ CPU target type <ul style="list-style-type: none"> <li>• C64 for Se-Series Gen2</li> <li>• C64+ for CSe-Series</li> <li>• SDK for Simulator for Windows</li> <li>• SIM for Simulator for Linux</li> </ul> </li> </ul>
version	Version number in format <code>x.y.z.w</code> : <ul style="list-style-type: none"> <li>▪ x: One byte in decimal format</li> <li>▪ y: One byte in decimal format</li> <li>▪ z: One byte in decimal format</li> <li>▪ w: One byte in decimal format</li> </ul>
name	Full name of the corresponding firmware module

Table 22: Meaning of the output parameters of csadm ListFiles

If the file is a cryptographic key ( `*.key` ), additionally the following information is displayed:

- First five bytes of the SHA-512 hash calculated over the modulus of the key



The file system of the device is case-sensitive.

### 4.8.3 LoadFile

This command loads a file (firmware module or other file) onto the working directory ( `FLASH` ) or the non-volatile directory ( `NVRAM` ) of the device.

If the file is a firmware module ( `*.mtc` ) the MMC signature of the firmware module will be checked. If it can't be successfully verified the loading of the module will be rejected.

If the firmware module is encrypted the device looks for the private part of the Firmware Encryption Key and tries to decrypt the module. If the firmware module could be successfully decrypted, it is encrypted with the Master Key before it is stored in the working directory

\FLASH (with extension `.msc`). Otherwise, it is stored by default in the \FLASH directory with the extension `.emc`, and will be decrypted later while loading the Firmware Encryption Key. Firmware modules with the `.emc` file extension cannot be executed. For details on how to load an encrypted firmware module into the device.



For security reasons, some files and file types cannot be loaded into the CryptoServer. Any try to load a file/file type, which is listed below, into the CryptoServer will cause an error.

- `prod*.key`
- `init*.key`
- `mdlsig*.key`
- `sens`
- `*.mt_`
- `*.sl_`
- `*.msc`
- `*.sys`
- `*.emc`
- `*.db`
- `*.log`



If the given file already exists, it will be replaced.

A loaded/replaced firmware module does not become active until the device has been restarted.

**Syntax**

```
csadm [Dev=<device>] <Authentication> LoadFile=<file>[,<dir>]
```

**Authentication**

Permission level 2 in the user group 6

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<file>	Filename (eventually with path)
<dir>	Device directory, where the file should be stored ( <code>FLASH</code> or <code>NVRAM</code> ). If omitted, the <code>\FLASH</code> directory is used as default.

**Example**

- `csadm LogonSign=ADMIN,d:\keys\myKey.key#ask  
LoadFile=exmp_dbg_1.0.2.3.mtc`
- `csadm dev=PCI:0 LogonPass=sven,swordfish  
LoadFile=usage.cnt,NVRAM`

**Output**

Upon successful execution of the command, no output is given.



If the filename of a firmware module contains an underscore character ('\_') the rest of the name up to the file extension is stripped before loading (for example, `exmp_dbg.mtc` will be loaded as `exmp.mtc`, `adm_1.0.0.1.mtc` will be loaded as `adm.mtc`).

The length of the file name (without directory) may not exceed 15 characters (here only those characters count that will not be stripped, i.e., characters before any underscore character '\_').

If no path is given with the <file> parameter, the file from the current directory is used. If the file to be loaded is stored in another directory (e.g., on an USB flash drive) the respective path must be given.



The file system of the device is case-sensitive.

Filenames of firmware modules (with extension `*.mtc`) are converted to lower case letter before being loaded onto the device.

If several files shall be loaded in one step, the last part of the command ( `LoadFile==<file>[,<dir>]` ) has to be repeated for every wanted file.

#### 4.8.4 LoadPkg

This command provides to the user a comfortable and easy-to-use possibility to load or update a set of several firmware modules and/or files into the device in only one step. It can replace a series of succeeding csadm commands (see example below).

The `LoadPkg` command loads the contents of the given package file (archive `*.mpkg`) into the device. Depending on the given command-line flags the load procedure can also perform a Clear if needed (`csadm clear=INIT`).

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; [SignedLicenseFile=&lt;license_file&gt;] ..... LoadPkg=&lt;package&gt;[,&lt;flags&gt;[,&lt;password&gt;]]</code>
---------------	--

<b>Authentication</b>	Permission level 2 in the user group 6
-----------------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<license_file>	Signed license file to be loaded in addition to the firmware package
<package>	Input package file containing device firmware modules and files (filename must have extension <code>*.mpkg</code> )
<flags>	Flags can be added (+) if this makes sense. <ul style="list-style-type: none"> <li><code>NoClear</code> (default) The device will not be cleared prior to package loading</li> <li><code>ForceClear</code> The device is cleared prior to package loading in any case, i.e., a <code>csadm ... Clear=INIT</code> command is automatically sent to the device. As a result, all sensitive data on the device is deleted before the contents of the package file is loaded.</li> </ul>
<password>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . <ul style="list-style-type: none"> <li>for hidden password entry the string 'ask' is used. We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>



**Example**

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadPkg=firmware.mpkg,NoClear
```

**Output**

Upon successful execution of the command, information about all actually performed steps and changes in status, mode or firmware module versions is returned.



By default, only modules are loaded that either are available on the device and are available with a higher version in the package or that are not available on the device but are available in the package. This is the usual upgrade situation.

To remove modules that are available on the device but not in the package or to downgrade one or several modules from a higher version on the device to a lower version in the package, use the `ForceClear` option. Attention: Using the `ForceClear` option removes all sensitive data from the device because a `csadm ... Clear=INIT` command is automatically sent to the device.



For CryptoServer 4.45 and later, the following applies:

The implementation of cryptographic primitives has been optimized for, and harmonized across, Utimaco's HSM platforms and host components, resulting in a general performance increase of cryptographic operations. The exact improvements depend on algorithm, key size, size of data to be encrypted/signed, and HSM model. The improvements are highest for short-running operations like AES encryption of small blocks of data. This new common cryptographic library is delivered as CRYPT firmware module. In a firmware package, this CRYPT firmware module replaces the following modules that were previously handled as separate modules: AES, HASH, DSA, VRSA, LNA, ECDSA, ECA, POST and ASN.1. The output of a `csadm LoadPkg` command therefore shows the CRYPT module instead of separate AES, HASH, etc. modules

The example outputs may differ from the output on your CryptoServer with respect to the listed firmware modules and/or their versions depending on the installed firmware package and its version.

**Example 1** (for CryptoServer 4.45 and later)

The example output may differ from the output on your CryptoServer with respect to the listed firmware modules and/or their versions depending on the installed firmware package and its version. The firmware, which is currently stored on the device shall be completely replaced by the firmware (and other files) contained in a `SecurityServer-Simulator-99.99.99.99-windows.mpkg` package file. In this case, a Clear should be performed as part of the `LoadPkg` command, i.e., the `ForceClear` flag must be set. During command execution information about all currently performed steps, the state and the mode of the device is displayed:

```

csadm Dev=3001@127.0.0.1 LogonSign=ADMIN,admin.key LoadPkg=SecurityServer-
Simulator-99.99.99.99-windows.mpkg,ForceClear
I: Reading package...
I: Going to delete all files/modules inside the CryptoServer (perform Clear)
I: > load file adm_3.1.2.0_c50.mtc...
I: > load file cmds_3.8.4.1_c50.mtc...
I: > load file crypt_2.2.1.0_c50.mtc...
I: > load file cxi_2.4.11.1_c50.mtc...
I: > load file db_2.0.0.2_c50.mtc...
I: > load file mbk_2.5.1.1_c50.mtc...
I: > load file ntp_1.2.1.1_c50.mtc...
I: > load file pp_1.4.1.2_c50.mtc...
I: > load file sc_1.2.0.7_c50.mtc...
I: > load file smos_5.6.4.1_c50.mtc...
I: > load file stun_0.0.0.1_c50.mtc...
I: > load file util_3.0.7.1_c50.mtc...
I: loaded 12/12 files in package
I: Restarting CryptoServer
Package SecurityServer-Simulator-99.99.99.99-windows.mpkg successfully loaded

```

#### Example 2 (earlier than CryptoServer 4.45)

The firmware, which is currently stored on the device shall be completely replaced by the firmware (and other files) contained in a `cxi_1.3.1.3.mpkg` package file. In this case, a Clear should be performed as part of the `LoadPkg` command, i.e., the `ForceClear` flag must be set. During command execution information about all currently performed steps, the state and the mode of the device is displayed:

```

csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadPKG=d:
\temp\cxi_1.3.1.3.mpkg,ForceClear
I: Reading package...
I: Perform authentication and create session
I: Going to delete all files/modules inside the CryptoServer (perform Clear)
I: Load file smos_3.0.1.0_c86.mtc
I: Load file util_3.0.0.1_c86.mtc
I: Load file cmds_3.0.0.2_c86.mtc
I: Load file adm_3.0.0.2_c86.mtc
I: Load file db_1.1.2.2_c86.mtc
I: Load file pp_1.2.3.0_c86.mtc
I: Load file sc_1.2.0.0_c86.mtc
I: Load file asn1_1.0.3.3_c86.mtc
I: Load file hash_1.0.6.0_c86.mtc
I: Load file aes_1.0.4.1_c86.mtc
I: Load file vdes_1.0.3.0_c86.mtc
I: Load file lna_1.1.0.0_c86.mtc
I: Load file vrsa_1.1.0.1_c86.mtc
I: Load file eca_1.1.1.0_c86.mtc
I: Load file ecdsa_1.0.1.0_c86.mtc

```

```
I: Load file dsa_1.0.0.0_c86.mtc
I: Load file mbk_2.1.2.3_c86.mtc
I: Load file hce_1.0.0.0_c86.mtc
I: Load file cxi_1.3.1.3_c86.mtc
I: Restarting CryptoServer
Package d:\temp\cxi_1.3.1.3.mpkg successfully loaded
```

### Example 3 (earlier than CryptoServer 4.45)

In the following example, only some firmware modules are upgraded (AES, CMDS), modules with higher version on the device remain unchanged and others that have not been yet available on the device are loaded.

```
csadm LogonSign=ADMIN,ADMIN.key LoadPkg=fw1.mpkg
I: Reading package...
I: Perform authentication and create session
I: Retrieving file list from CryptoServer
I: Load file exmp_1.0.1.0.mtc
I: Load file pkcs11_1.0.6.1.mtc
I: Load file cmds_3.0.0.2_c86.mtc
I: Load file aes_1.0.5.0_c86.mtc
I: Restarting CryptoServer
Package fw1.mpkg successfully loaded
```

## 4.8.5 DeleteFile

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

The following files or file types cannot be deleted due to security restrictions:

- prod\*.key
- init\*.key
- mdlsig\*.key
- sens
- \*.sys
- \*.log
- \*.scf



This command can also be used to delete firmware modules ( `*.msc` files). However, the set of backup copies of the basic firmware modules (system firmware modules `*.sys`) cannot be deleted.

Even if already deleted from the working directory, a firmware module is still running (for Se-Series Gen2: in SD-RAM memory; for CSe-Series: in DDR2 RAM memory). The firmware module only becomes inactive after the device has been restarted.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

The following is an example for a correct csadm command entry in the Microsoft PowerShell:

```
csadm [Dev=<device>] LogonSign="<user>,<keyspec>" <command>
```

### Syntax

```
csadm [Dev=<device>] <Authentication> DeleteFile=[<dir>\]<file>
```

### Authentication

Permission level 2 in the user group 6 (02000000)

Parameter	Description
<device>	Device address of the device or LAN device This parameter can be omitted if the device address has been set as the CRYPTOSERVER environment variable
<file>	Filename
<dir>	Device directory ( <code>FLASH</code> or <code>NVRAM</code> ). If omitted, the working directory (FLASH) is used as default.

### Example

Example 1:

```
csadm LogonPass=sven,swordfish DeleteFile=exmp.msc
```

Example 2:

```
csadm LogonSign=sm,:cs2:cjo:USB0
```

```
DeleteFile=FLASH\eeexmp.msc
```

Example 3: Delete a file in the non-volatile RAM

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 DeleteFile="NVRAM\
\foo.txt"
```

**Output**

Upon successful execution of the command, no output is given.



The file system of the device is case-sensitive.

## 4.8.6 Pack

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

**Syntax**

```
csadm Pack=<directory>
```

**Authentication**

Permission level 2 in the user group 6 (02000000)

Parameter	Description
<directory>	Input directory containing device firmware modules and files

**Example**

```
csadm Pack=C:\firmware\release-1.0.0
```

**Output**

Upon successful execution of the command, a success message is returned.  
 Package C:\firmware\release-1.0.0.mpkg created

The resulting `.mpkg` package file is stored in the same directory as the given input directory. Any existing file with the same name is overwritten.

## 4.8.7 Unpack

This command extracts all files contained in a device package file `*.mpkg`. The files will be extracted in a new directory which will have the same name as the given package file but without the `.mpkg` extension.

<b>Syntax</b>	csadm Unpack=<package>
---------------	------------------------

<b>Authentication</b>	Permission level 2 in the user group 6 (02000000)
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<package>	Input package file ( *.mpkg ) containing device firmware modules and files

<b>Example</b>	csadm Unpack= C:\firmware\release-1.0.0.mpkg
----------------	--

<b>Output</b>	Upon successful execution of the command, a success message is returned. Package C:\firmware\release-1.0.0.mpkg unpacked successfully The resulting .mpkg package file is stored in the same directory as the given input directory. Any existing file with the same name is overwritten.
---------------	---

The resulting directory `firmware` containing the extracted firmware modules and files will be created in the same directory where the given `firmware.mpkg` package file is located.

If a directory of that name already exists, the extraction is denied.

### 4.8.8 ListPkg

This command lists the content of a device package file (\* .mpkg ), i.e., the filenames of all contained files. If firmware modules are contained in the package the version numbers of the modules and long names are also listed.

<b>Syntax</b>	csadm ListPkg=<package>
---------------	-------------------------

<b>Authentication</b>	Permission level 2 in user group 6 (02000000)
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<package>	Input package file ( * .mpkg ) containing device firmware modules and files

**Example**

Example 1 for CryptoServer 4.45:

```
csadm ListPkg=SecurityServer-Simulator-4.45.4.0-windows.mpkg
```

Example 2 for prior to CryptoServer 4.45:

```
csadm ListPkg= C:\firmware\release-1.0.0.mpkg
```

<b>Output</b>	<p>Upon successful execution of the command, the content of the package file is returned.</p> <p>The example output may differ from the output on your CryptoServer with respect to the listed firmware modules, their types and/or their versions depending on the installed firmware package and its version.</p> <p>For CryptoServer 4.45 and later, the following applies:</p> <p>The implementation of cryptographic primitives has been optimized for, and harmonized across, Utimaco's HSM platforms and host components, resulting in a general performance increase of cryptographic operations. The exact improvements depend on algorithm, key size, size of data to be encrypted/signed, and HSM model. The improvements are highest for short-running operations like AES encryption of small blocks of data. This new common cryptographic library is delivered as CRYPT firmware module. In a firmware package, this CRYPT firmware module replaces the following modules that were previously handled as separate modules: AES, HASH, DSA, VRSA, LNA, ECDSA, ECA, POST and ASN.1. The output of a <code>csadm ListPkg</code> command therefore shows the CRYPT module instead of separate VRSA etc. modules</p> <p>Output for the given example 1:</p> <pre>Archive SecurityServer-Simulator-4.45.4.0-windows.mpkg contains: 1. - adm 3.1.0.1_c50.mtc [SDK] (Administration Module SDK version 3.1.0.1) 2. - cmds 3.7.2.1_c50.mtc [SDK] (Command Scheduler (SDK) version 3.7.2.1) 3. - crypt_2.1.1.0_c50.mtc [SDK] (CRYPT Module SDK version 2.1.1.0) 4. - cxi 2.4.6.5_c50.mtc [SDK] (Crypto eXtended Interf.SDK version 2.4.6.5) 5. - db 2.0.0.0_c50.mtc [SDK] (Database module (SDK) version 2.0.0.0) 6. - mbk 2.5.0.1_c50.mtc [SDK] (Master Backup Key Module SDK version 2.5.0.1) 7. - ntp 1.2.1.1_c50.mtc [SDK] (TimeAdjust Module SDK version 1.2.1.1) 8. - pp 1.4.1.0_c50.mtc [SDK] (PIN pad Driver (SDK) version 1.4.1.0) 9. - sc 1.2.0.6_c50.mtc [SDK] (Smartcard module (SDK) version 1.2.0.6) 10. - smos 5.6.4.0_c50.mtc [SDK] (Operating System SDK version 5.6.4.0) 11. - util 3.0.7.0_c50.mtc [SDK] (Utility Module SDK version 3.0.7.0)</pre> <p>Output for example 2 prior to CryptoServer 4.45:</p>
---------------	--



```
Archive release-1.0.0.mpkg contains:
1. - adm_1.0.1.1.mtc (Administration Module version 1.0.1.1)
2. - asn1_1.0.1.2.mtc (ASN1 Module version 1.0.1.2)
3. - cmds_1.0.6.0.mtc (Command Scheduler version 1.0.6.0)
4. - db_1.0.1.1.mtc (Database module version 1.0.1.1)
5. - hash_1.0.1.0.mtc (Hash Module version 1.0.1.0)
6. - lna_1.0.3.0.mtc (Long Number Arithmetic version 1.0.3.0)
7. - smos_1.0.3.5.mtc (SMOS Operating System version 1.0.3.5)
8. - util_1.0.5.0.mtc (Utility Module version 1.0.5.0)
9. - vdes_1.0.0.3.mtc (DES Module version 1.0.0.3)
10. - vrsa_1.0.4.0.mtc (RSA Module version 1.0.4.0)
```

### 4.8.9 CheckPkg

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

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

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CheckPkg=&lt;package&gt;[,&lt;password&gt;]</code>
---------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<package>	Input package file ( <code>*.mpkg</code> ) containing device firmware modules and files
<password>	Root password of the LAN device This parameter is only relevant if the driver commands <code>Reset</code> , <code>Restart</code> and <code>ResetToBL</code> are password protected, which depends on the settings in the LAN device configuration file <code>csxlan.conf</code> . <ul style="list-style-type: none"> <li>For hidden password entry the string 'ask' is used. We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm CheckPkg=firmware.mpkg</code>
----------------	---

<b>Output</b>	<p>Upon successful execution of the command, information about differences between loaded firmware and firmware contained in given package file is returned.</p> <pre>I: Reading package... I: Retrieving file list from CryptoServer Package d:\temp\cxi_1.3.1.3.mpkg successfully checked</pre>
---------------	---

### 4.8.10 ModuleInfo

This command shows the module information of a firmware module. MTC, MMC or raw binary firmware module files can be given as an input file.

<b>Syntax</b>	<code>csadm ModuleInfo=&lt;file&gt;</code>
---------------	--

<b>Authentication</b>	Permission level 2 in user group 6 (02000000)
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<file>	Firmware module ( *.mtc, *.mmc, *.out, *.dll or *.so )

<b>Example</b>	<code>csadm ModuleInfo=cmds.mtc</code>
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the module information is returned.</p> <pre>Module          'CMDS' ID              83 Version         3.5.2.0 Name            'Command Scheduler' Module type     'CS2-COFF' MMC header vers. 1.1.0.0 Target CPU type 'C64x' Hardware type   'C50'  Sig. algorithm  RSA Hash algorithm  SHA512 Key hash        -2B975A1976-</pre>
---------------	---



If multiple firmware modules shall be processed in one step, the last part of the command ( `ModuleInfo=<file>` ) has to be repeated for each firmware module.

The meaning of the output information is as follows:

<i>Field</i>	<i>Description</i>
Module	Name of the firmware module, e.g., 'CMDS'
ID	Unique identifier (ID) also called function code (FC) of the firmware module, e.g., 83, for the CMDS firmware module. This is hexadecimal number without a leading 0x.
Version	Version number of the firmware module, e.g., 3.3.0.1
Name	Extended (full) name of the firmware module, e.g., 'Command Scheduler'
Module type	Type of the raw binary file that is contained in MTC/MMC: <ul style="list-style-type: none"> <li>CS2-COFF if binary is compiled for device hardware,</li> <li>SDK-DLL if binary is compiled for device Simulator for Windows</li> <li>SDK-SO if binary is compiled for device Simulator for Linux</li> <li>unknown if the input file is a binary file.</li> </ul>
MMC header vers.	Version of MMC header (will be returned only for MTC or MMC file): 1.1.0.0: current version (Note: SMOS version $\geq$ 2.1.0.0 is needed for interpretation of MMC header)
Target CPU type	CPU type of target platform: <ul style="list-style-type: none"> <li>C64+ for CSe-Series</li> <li>C64x for Se-Series Gen2</li> <li>SDK for device simulator for Windows</li> <li>SIM for device simulator for Linux</li> </ul>
Hardware type	Hardware type/software architecture type: <ul style="list-style-type: none"> <li>C50 for Se-Series Gen2 (bootloader version <math>\geq</math> 5.0.0.0)</li> <li>C57 for CSe-Series (bootloader version <math>\geq</math> 4.0.0.0)</li> </ul>
Sig. algorithm	RSA Signature algorithm used for the generation of Utimaco's Module Signature Kek.

<b>Field</b>	<b>Description</b>
Hash algorithm	SHA512 Hash algorithm used for calculating the hash value of the Module Signature Key
Key hash	Hash value of the RSA Module Signature Key

Table 23: Meaning of ModuleInfo fields

### 4.8.11 Clear

The `Clear` command erases all sensitive data from the device. Only the data of users with public key authentication mechanisms remain in the user database, depending on the used variant of the command.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] [&lt;Authentication&gt;] Clear[=&lt;flag&gt;]</code>
---------------	---

<b>Authentication</b>	Permission 2 in the user group 6 for <code>Clear=INIT</code> (02000000) No authentication for <code>Clear=DEFAULT</code> , an external erase must have been initiated just before.
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

Parameter	Description
<flag>	<p>Command flag which can be set to:</p> <ul style="list-style-type: none"> <li>▪ <b>INIT</b> The following sensitive data is deleted: <ul style="list-style-type: none"> <li>• Master Key. The deletion of the Master Key automatically makes all other keys and sensitive data stored on the device unusable because they can no longer be decrypted without the Master Key.</li> <li>• HSM Authentication Key ( <code>authkey.db</code> ), if available</li> <li>• Firmware Encryption Key, if available</li> <li>• The audit log signature key ( <code>auditkey.db</code> ), if available</li> <li>• Master Backup Key (MBK), if available</li> <li>• All firmware modules ( <code>*.msc</code> files) except for the bootloader code and the system firmware modules ( <code>*.sys</code> files)</li> <li>• Signed configuration file <code>cmds.scf</code> , if available</li> <li>• All users in the user database using a HMAC password</li> </ul> <p>The following items are not deleted:</p> <ul style="list-style-type: none"> <li>• Public parts of the Production Key, Default Administrator Key, Module Signature Key and Alternative Module Signature Key</li> <li>• Bootloader code and system firmware modules ( <code>*.sys</code> )</li> <li>• Alarm state file (alarm.sens)</li> <li>• Audit log file(s) ( <code>audit*.log</code> )</li> <li>• Audit log configuration file ( <code>audit.cfg</code> )</li> <li>• Any Signed License File ( <code>*.slf</code> ), if present</li> <li>• Bootloader configuration file ( <code>bl.cfg</code> )</li> <li>• All users in the <code>user.db</code> user database using a public key authentication mechanism, e.g., RSA</li> <li>• A second copy of the public part of the Default Administrator Key ( <code>init.key</code> file).</li> </ul> <p>This copy can neither be changed nor deleted, but it is used to restore the default administrator user ADMIN with his Default Administrator Key as authentication token (using the <code>csadm Clear = DEFAULT ( ClearFactoryDefaults )</code> command or <b>CAT &gt; Manage &gt; Clear to Factory Settings</b>).</p> <p>Consider that the above list differs from the items that are deleted due to an alarm.</p> </li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>▪ <b>DEFAULT</b> This restores the factory default settings and resets the device into its delivery state. The Master Key of the device and other sensitive data are erased. The user database is deleted and recreated with the default user ADMIN and the corresponding Default Administrator Key. In addition to the files that the <code>csadm Clear=INIT</code> command deletes, the <code>csadm Clear=DEFAULT</code> command or <b>CAT &gt; Manage &gt; Clear to Factory Settings</b> deletes, for example, a second copy of the public part of the Default Administrator Key (<code>init.key</code> file). This copy is used to restore the default administrator user ADMIN with his Default Administrator Key as authentication token.</li> </ul> <p>If the command parameter <code>&lt;flag&gt;</code> is omitted, the INIT version of the command is executed by default.</p>

<b>Example</b>	<pre>csadm LogonSign=ADMIN,c:\keys\myKey.key Clear=INIT csadm Clear=DEFAULT</pre>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---



With the command `Clear=DEFAULT` the device is reset to factory defaults, i.e. the device can be restored in case the user has locked out himself from administration (for example, lost smartcard or forgotten password).

The command can only be executed if the user has proven physical presence by triggering an external erase (which can only be performed locally, i.e., directly at the device).

## 4.9 Commands for Firmware Packaging

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

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

More details about the concepts of MTCs and package files can also be found in *Firmware Module Management* in the *CryptoServer Administration Manual*.

All firmware modules are packed into a signed data container before they are loaded into the device.

## Module Transport Container - MTC

MTC Header	MMC Header	Binary (*out, *.dll, *.so)	MMC Signature (Module Signature Key)
------------	------------	-------------------------------	---

This signature is mandatory and is intended to be created by the author of the firmware module. The device - and anyone else - is able to check the authenticity of the firmware module's origin.

When a firmware module is loaded into the device the operating system (SMOS) checks the signature in the following order:

1. First SMOS tries to verify the signature with the public part of the Module Signature Key of Utimaco IS GmbH, which is stored in every device .
2. If the verification fails and if an Alternative Module Signature Key has been loaded by the customer, see [LoadAltMdlSigKey \(p. 184\)](#), the operating system also tries to verify the signature with the Alternative Module Signature Key. By this means, a customer is able to load a self-programmed and self-signed firmware module.

Optionally, a firmware module can also be encrypted with the public part of a Firmware Encryption Key. In this case, the private part of the Firmware Encryption Key has to be loaded into the device as well, so that the operating system is able to decrypt the firmware module, see [LoadFWDecKey \(p. 183\)](#).

The structure and signature of a firmware module container can be checked manually (outside the device ) using the `csadm VerifyMTC` command, see [VerifyMTC \(p. 179\)](#).

The following command group contains internal functions of csadm. No connection to a device is established. To see the list of these commands in the in-tool `csadm help` , perform the `csadm More` command in the command-line.



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a csadm command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

### 4.9.1 MakeMTC

This command builds the Module Transport Container file (MTC, \*.mtc) from a raw binary firmware module (\*.out, \*.dll, \*.so) or from a Module Manufacturer Container file (MMC, \*.mmc).



Firmware modules created by Utimaco IS GmbH are always signed with the private part of the Manufacturer's Module Signature Key. As the corresponding public key part is loaded into every CryptoServer, these firmware modules can be loaded without additional measures.

Firmware modules created (programmed) by the customer have to be signed with an Alternative Module Signature Key, which is also created by the customer. To be able to load a self-signed firmware module into the device, the public part of the Alternative Module Signature Key has to be stored inside the device.



From SecurityServer 4.00 only signing of firmware modules using SHA-512 is supported.

Firmware modules that have been built using the administration tool csadm provided with device SDK 3.30 (or previous) do not need to be recompiled but must be resigned with `csadm MakeMTC` (csadm version 2.0.0.1 and higher) to run on device with SecurityServer 4.00 resp. CryptoServer SDK 4.00 firmware.

#### Syntax

- If `<file>` is a raw binary firmware module (\*.out, \*.dll, \*.so):
  - `csadm [Model=<model>] MMCSignKey=<keyspec> [FWEncKey=<keyspec>] MakeMTC=<file>`
  - If the Module Signature Key is stored in a CryptoServer device:
 

```
csadm [Dev=<device>] <Authentication>
[Model=<model>] MMCSignKey=<keyspec>
[FWEncKey=<keyspec>] MakeMTC=<file>
```
- If `<file>` is a Module Manufacturer Container file (MMC, \*.mmc):
  - `csadm [Model=<model>] [MMCSignKey=<keyspec>] [FWEncKey=<keyspec>] MakeMTC=<file>`
  - If the Module Signature Key is stored in a CryptoServer device:
 

```
csadm [Dev=<device>] <Authentication>
[Model=<model>] [MMCSignKey=<keyspec>]
[FWEncKey=<keyspec>] MakeMTC=<file>
```



<b>Authentication</b>	<p>Permission 2 in the user group 0 if the Module Signature Key is stored in the cryptographic key database, <code>CXIKKEY.db</code></p> <p>Additionally, the <code>CXI_GROUP</code> user attribute of the user(s) authenticating the command has to match the <code>CXI_GROUP</code> attribute of the Module Signature Key.</p>
-----------------------	--

Parameter	Description
<device>	<p>Device address</p> <p>This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.</p>
<model>	<p>Hardware type</p> <p>Note that the &lt;model&gt; entry is case-sensitive.</p> <p>c50: Se-Series Gen2 (SMOS ≥ 5.0.0.0)</p> <p>c57: CSe-Series (SMOS ≥ 4.0.0.0)</p>
<keyspec>	<p><code>MMCSignKey</code> = private part of Module Signature Key</p> <p><code>FWEncKey</code> = public part of Firmware Encryption Key</p> <ul style="list-style-type: none"> <li>If the keyfile is stored on a smartcard  <code>&lt;keyfile&gt;/&lt;smartcard specifier&gt;</code>,  for example, <code>my_md1_sign.key/:cs2:cjo:USB0</code> </li> <li><code>&lt;keyfile&gt;[#password]</code>,  for example, <code>my_md1_sign.key,#ask</code>  password is needed in case of an encrypted keyfile.  If no password is given (or the password is replaced by the string 'ask'), hidden password entry will be performed for encrypted keyfiles, which we strongly recommend. </li> <li>If the Module Signature Key is stored in the cryptographic key database, <code>CXIKKEY.db</code>, of a (LAN) device:  Dev: <code>&lt;keyname&gt;[,&lt;key_spec(version)&gt;]</code> ,  for example, <code>Dev:CS_MSK,1</code>  Dev: references the device address defined in the parameter Dev=&lt;device&gt; or in the environment variable CRYPTOSERVER. Do not change this string.  The <code>key name</code> is the name of the keyfile without file extension *.key. It should be followed by the key specifier <code>key_spec(version)</code>, if any was defined on key generation. </li> <li>If &lt;file&gt; is a raw binary firmware module (<code>*.out</code>, <code>*.dll</code>, <code>*.so</code>), <code>MMCSignKey</code> is mandatory. If &lt;file&gt; is a Module Manufacturer Container file (MMC, <code>*.mmc</code>), <code>MMCSignKey</code> is optional.</li> </ul>
<file>	Firmware module ( <code>*.out</code> , <code>*.dll</code> , <code>*.mmc</code> , <code>*.so</code> )

<b>Example</b>	<p>Example 1: Build an MTC file by using a Module Signature Key stored in an encrypted keyfile.  <code>csadm Model=c50 MMCSignKey=c:\keys\my_md1_sign.key,#ask  MakeMTC=c:\firmware\exmp.out</code></p> <p>Example 2: Build an MTC file by using a Module Signature Key stored in a CryptoServer LAN device.  <code>csadm Dev=112.111.1.13 LogonPass=CSUser,ask  LogonPass=CSUser1,ask Model=c57 MMCSignKey=Dev:CS_MSK,1  MakeMTC=C:\firmware\exmp.out</code></p> <p>If the device address has been set as the environment variable CRYPTOSERVER:  <code>csadm LogonPass=CSUser,ask LogonPass=CSUser1,ask Model=c57  MMCSignKey=Dev:CS_MSK,1 MakeMTC=C:\firmware\exmp.out</code></p>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

## 4.9.2 RemoveMTC

A Module Transport Container (\*.mtc file) contains a Module Manufacturer Container (MMC), which contains a raw binary firmware module.

If an \*.mtc file is the input file for the `csadm RemoveMTC` command, this command copies the therein contained MMC into an \*.mmc file and the therein contained raw binary firmware module into an \*.out, \*.dll or \*.so file.

If an \*.mmc file is the input file for the `csadm RemoveMTC` command, this command copies the therein contained raw binary firmware module into an \*.out, \*.dll or \*.so file.

The output file(s) are created in the directory of the input \*.mtc or \*.mmc file. If these output file(s) already exist, they are overwritten without further notice. The input \*.mtc or \*.mmc file remains unchanged.

Performing the `csadm RemoveMTC` command is only necessary if the firmware module should be re-signed with a customer-generated Alternative Module Signature Key. In this case, the public part of this key has to be loaded into the device as Alternative Module Signature Key.

<b>Syntax</b>	<code>csadm RemoveMTC=&lt;file&gt;</code>
---------------	---

Parameter	Description
<file>	Firmware module as MTC ( *.mtc ) or MMC ( *.mmc ) If the <code>csadm RemoveMTC</code> command shall be applied to multiple MTCs or MMCs in one step, the last part of the command ( <code>RemoveMTC=&lt;file&gt;</code> ) has to be repeated for each firmware module.

<b>Example</b>	<p>Example 1: <code>csadm RemoveMTC=C:\firmware\exmp.mtc</code></p> <p>Example 2: <code>csadm RemoveMTC=C:\firmware\exmp.mmc</code></p> <p>Example 3: <code>csadm RemoveMTC=exmp.mtc RemoveMTC=exmp02.mtc</code></p> <p>Example 4: <code>csadm RemoveMTC=exmp.mmc RemoveMTC=exmp02.mmc</code></p>
----------------	---

<b>Output</b>	<p>Upon successful execution of the command, the following output is returned for the given examples:</p> <p>Example 1: exmp.mtc:              removing MTC ... OK              removing MMC ... OK</p> <p>Example 2: exmp.mmc:              removing MMC ... OK</p> <p>Example 3: exmp.mtc:              removing MTC ... OK              removing MMC ... OK          exmp02.mtc:              removing MTC ... OK              removing MMC ... OK</p> <p>Example 4: exmp.mmc:              removing MMC ... OK          exmp02.mmc:              removing MMC ... OK</p>
---------------	--

### 4.9.3 VerifyMTC

This command verifies the header of the Module Transport Container MTC ( \*.mtc ) and the signature of the therein contained Module Manufacturer Container MMC ( \*.mmc ).

<b>Syntax</b>	<code>csadm MMCPubKey=&lt;keyspec&gt; VerifyMTC=&lt;file&gt;</code>
---------------	---

<b>Parameter</b>	<b>Description</b>
<keyspec>	Public part of the Module Signature Key <ul style="list-style-type: none"> <li>▪ Smartcard specifier, e.g. :cs2:cjo:USB0</li> <li>▪ Keyfile name, e.g., mykey.key.</li> </ul>
<file>	Firmware module as MTC ( *.mtc ) or MMC ( *.mmc ) If multiple MTCs shall be verified in one step, the last part of the command ( VerifyMTC=<file> ) has to be repeated for each firmware module.

<b>Example</b>	<p>Example 1:</p> <pre>csadm MMCPubKey=:cs2:cjo:USB0 VerifyMTC=c:\firmware\exmp.mtc</pre> <p>Example 2:</p> <pre>csadm MMCPubKey=:cs2:cjo:USB0 VerifyMTC=c:\firmware\exmp.mmc</pre>
----------------	---

<b>Output</b>	<p>Upon successful execution of the command, the following output is returned for the given examples:</p> <p>Example 1:</p> <pre>Verifying and Removing MTC ...OK Verifying MMC ... OK, signature type = 1</pre> <p>Example 2:</p> <pre>Verifying MMC ... OK, signature type = 1 signature type = 1 means that PKCS#1 padding is used.</pre>
---------------	--

#### 4.9.4 VerifyPkg

This command verifies the signature of the Module Transport Container MTC ( \*.mtc ) of each firmware module that is contained in the given package file (archive \*.mpkg).

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

<b>Syntax</b>	<code>csadm MMCPubKey=&lt;keyspec&gt; VerifyPkg=&lt;package&gt;</code>
---------------	--

Parameter	Description
<keyspec>	<p>Public part of the Module Signature Key</p> <ul style="list-style-type: none"><li>▪ If the key is stored on a smartcard <code>&lt;keyfile&gt;/&lt;smartcard specifier&gt;</code>, for example, <code>my_md1_sign.key/:cs2:cjo:USB0</code></li><li>▪ If the key is stored in a keyfile <code>&lt;keyfile&gt;[#&lt;password&gt;]</code>, for example, <code>my_md1_sign.key,#ask</code> password is needed in case of an encrypted keyfile. If no password is given (or the password is replaced by the string 'ask'), a hidden password entry will be performed for encrypted keyfiles, which we strongly recommend.</li></ul>
<package>	Input package file containing device firmware modules and files ( * .mpkg )

<b>Example</b>	<code>csadm MMCPubKey=mdl_sign_pub.key verifypkg=~/tmp/SecurityServer-CSe-Series-FIPS-4.31.0.01.mpkg</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, Information about the firmware module signature is returned.
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
	Verifying and Removing MTC ... OK
	Verifying MMC ... OK, signature type = 1
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK, signature type = 1	
Verifying and Removing MTC ... OK	
Verifying MMC ... OK,	

#### 4.9.5 RenameToVersion

This command verifies the signature of the Module Transport Container MTC ( `*.mtc` ) of each firmware module that is contained in the given package file (archive `*.mpkg` ) .

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

<b>Syntax</b>	<code>csadm RenameToVersion=&lt;file&gt;</code>
<b>Parameter</b>	<b>Description</b>
<file>	firmware module (*.mtc, *.out, *.dll or *.so)
<b>Example</b>	<code>csadm RenameToVersion=C:\modules\vdes.mtc</code> For example, the file C:\modules\vdes.mtc is renamed to C:\modules\vdes_1.0.1.0.mtc depending on the current firmware module version
<b>Output</b>	Upon successful execution of the command, no output is given.

#### 4.9.6 LoadFWDecKey

This command loads a private RSA key into the device which is used to decrypt encrypted firmware modules (private part of Firmware Encryption Key).

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; LoadFWDecKey=&lt;keyspec&gt;</code>
<b>Authentication</b>	Permission level 2 in user group 5 (02000000)
<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<keyspec>	Key specifier of Firmware Encryption Key. May be either a path name to a file containing the private key, or a smartcard descriptor if the key is stored as two XOR-halves on back-up smartcards.
<b>Example</b>	<code>csadm LogonSign=ADMIN,d:\keys\myKey.key ...LoadFWDecKey=C:\keys\fw_dec.key</code> <code>csadm LogonPass=sven,swordfish ... LoadFWDecKey=:cs2:cjo:USB0</code>
<b>Output</b>	Upon successful execution of the command, no output is given.

### 4.9.7 LoadAltMdlSigKey

This command loads the public part of the customer's Alternative Module Signature Key into the device. This key is later used to verify the authenticity of self-programmed firmware modules during the load process, which are signed by the customer with the private part of his Alternative Module Signature Key.

The minimum length of an Alternative Module Signature Key is 1024 bit for u.trust Anchor.

The minimum length of an Alternative Module Signature Key is 2048 bit for u.trust Anchor CC.

The minimum length of an Alternative Module Signature Key is 2048 bit for u.trust Anchor FIPS.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] &lt;Authentication&gt; LoadAltMdlSigKey=&lt;keyspec&gt;</code>
---------------	---

<b>Authentication</b>	Permission level 2 in user group 6 and 7 (22000000)
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<keyspec>	Key specifier of the public part of the customer's Alternative Module Signature Key

<b>Example</b>	<code>csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadAltMdlSigKey=MyMdlSig.key</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---



The key will be stored on the device as file `mdlsigalt.key`. If the key already exists, it will be overwritten.





Firmware modules created by Utimaco IS GmbH are always signed with the Utimaco Module Signature Key. Since the public part of this key is loaded into every device the operating software is always able to verify the signature on load of any firmware module which is created by Utimaco IS GmbH.

#### 4.9.8 SignConfig



This command is only available in csadm version 1.8.11 and later.

The `csadm SignConfig` command signs a self-created configuration file `cmds.cfg` (or any other file extension) with the Alternative Module Signature Key. The signed configuration file is specified by the file extension `*.scf`, and allows you to define specific firmware module configuration settings, for example, to disable some selected device functions. The syntax of the signed configuration file follows the syntax of configuration files.



The use of signed configuration files is supported for the CSe-Series and Se-Series Gen2 with the following min. versions of the firmware modules SMOS, ADM and CMDS:

For the CSe-Series:

ADM version 3.0.12.0  
CMDS version 3.2.0.2  
SMOS version 4.4.0.0

For the Se-Series Gen2:

ADM version 3.0.12.0  
CMDS version 3.2.0.2  
SMOS version 5.1.0.0

#### Syntax

```
csadm MdlSignKey=<keyspec> SignConfig=<file>
```

If the Module Signature Key is stored in a device:

```
csadm [Dev=<device>] <Authentication> MdlSignKey=<keyspec>  
SignConfig=<file>
```

<b>Authentication</b>	Permission level 2 in user group 6 and 7 (22000000)
-----------------------	---

Parameter	Description
<keyspec>	<p>Key specifier for the MdlSignKey which is the private part of the Module Signature Key or the Alternative Module Signature Key. Can be defined as:</p> <ul style="list-style-type: none"> <li>Smartcard specifier If the Module Signature Key or the Alternative Module Signature Key is stored on a smartcard, for example, <code>:cs2:cjo:USB0</code></li> <li>Keyfile specifier (keyfile name and, if needed, file path) If the Module Signature Key or the Alternative Module Signature Key is stored in a keyfile In case an encrypted keyfile is used, the correct password for the keyfile is required, i.e., <code>&lt;keyfile&gt;#password</code> . <code>password</code> is needed in case of an encrypted keyfile. If no password is given (or the password is replaced by the string 'ask'), hidden password entry will be performed for encrypted keyfiles, which we strongly recommend.</li> <li>If the Module Signature Key is stored in the cryptographic key database, <code>CXKEY.db</code> , of a (LAN) device: <code>Dev:&lt;keyname&gt;[,&lt;key_spec(version)&gt;]</code> , for example, <code>Dev:CS_MSK,1</code> <code>Dev:</code> references the device address defined in the parameter <code>Dev=&lt;device&gt;</code> or in the environment variable CRYPTOSERVER. Do not change this string. The keyname is the name of the keyfile without file extension *.key. It should be followed by the key specifier <code>key_spec(version)</code> , if any was defined on key generation.</li> </ul>
<file>	Name and storage location of the configuration file ( <code>cmds.*</code> ) to be signed

<b>Example</b>	<p>Sign a configuration file by using a Module Signature Key stored in a keyfile. <code>csadm MdlSignKey=D:\keys\myMDLSign.key SignConfig=C:\myConfig\cmds.cfg</code></p> <p>Sign a configuration file by using a Module Signature Key stored on a smartcard. <code>csadm MdlSignKey=:cs2:cjo:USB0 SignConfig=C:\myConfig\cmds.txt</code></p> <p>Sign a configuration file by using a Module Signature Key stored in a CryptoServer LAN device. <code>csadm Dev=192.168.0.1 LogoSign=SysAdmin,:cs2:cjo:USB0 MdlSignKey=Dev:CS_MSK,1 SignConfig=C:\myConfig\cmds.txt</code></p>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is given.
---------------	---

Please find detailed information on how to use a signed configuration file to disable selected device functions or to increase the authentication requirements for selected device functions in [Creating and Using the Signed Configuration File `cmds.scf`](#) (p. 222).

## 4.10 Commands for Administration of CryptoServer LAN

This command group explicitly addresses the CryptoServer LAN appliance itself; commands will not be forwarded to the mounted CryptoServer PCIe card. Instead they will be processed by the control module of the TCP-Server (`csxlan-daemon`) and responded in the same way a firmware module would respond to a command.

These commands are used to administrate a CryptoServer LAN remotely (for example, to get/set its configuration).



Since the commands of this group are not directed to any CryptoServer PCIe card but only to the CryptoServer LAN, the presence, mode or state of eventually underlying CryptoServers are not relevant for their performance. For the same reason, none of these commands have to be authenticated using the CryptoServer's authentication mechanism.

Nevertheless, some commands of this group are protected against unauthorized use with an own authentication mechanism: these commands have to be authenticated with the root password of the CryptoServer LAN, see also the [CryptoServer LAN V5 - Administration Manual](#) (p. 243).



Certain types of shell processes treat certain characters (commas, colons, semi-colons) differently. If the execution of a `csadm` command fails with an error message from the shell about an illegal parameter format, quoting parameter values may be necessary.

### 4.10.1 CSLGetConnections

This command lists all current connections to the LAN device.

Syntax
<code>csadm [Dev=&lt;device&gt;] CSLGetConnections</code>

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	csadm Dev=10.17.1.9 CSLGetConnections
----------------	---------------------------------------

<b>Output</b>	<p>Upon successful execution of the command, a list of all current LAN connections is returned.</p> <p>current connections to CryptoServer LAN</p> <pre># prot port address ----- 1 TCP 1131 193.168.4.122 2 TCP 2563 194.169.4.098 3 TCP 1137 195.170.4.122</pre>
---------------	--

<b>Parameter</b>	<b>Description</b>
#	Connection number
prot	Used protocol (either UDP or TCP)
port	Port on the administration computer used to open the connection
address	IP address of the administration computer

Table 24: Meaning of the output parameters of csadm CSLGetConnections



Up to 10,000 connections can be established to one LAN device simultaneously.

## 4.10.2 CSLScanDevices

The `CSLScanDevices` command returns the internal configuration of one or more LAN devices as set up in the configuration file `/etc/csxlan.conf` on each LAN device. If the command is called using the dedicated IP address (as `Dev=<device>`) of a LAN device, only the configuration of this single LAN device will be shown.

However, `CSLScanDevices` can also be used to scan the whole network for all LAN devices currently available. Therefore, this command has to be sent as Multicast Message by using

the special IP address `UDP:224.1.0.1` as device parameter ( `Dev= UDP:224.1.0.1` ). It will be responded by every LAN device which has been set up to respond to Multicast Messages.



`CSLScanDevices` does only find such LAN devices which have been configured to respond Multicast Messages

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLScanDevices[=&lt;timeout&gt;]</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the CRYPTOSERVER environment variable. <device> = <code>UDP:224.1.0.1</code> (Multicast) if the whole network shall be scanned.
<timeout>	Timeout in ms (default: 3000)

<b>Example</b>	<p>Sign a configuration file by using a Module Signature Key stored in a keyfile.  <code>csadm MdlSignKey=D:\keys\myMDLSign.key SignConfig=C:\myConfig\cmds.cfg</code></p> <p>Sign a configuration file by using a Module Signature Key stored on a smartcard.  <code>csadm MdlSignKey=:cs2:cjo:USB0 SignConfig=C:\myConfig\cmds.txt</code></p> <p>Sign a configuration file by using a Module Signature Key stored in a CryptoServer LAN device.  <code>csadm Dev=192.168.0.1 LogoSign=SysAdmin,:cs2:cjo:USB0 MdlSignKey=Dev:CS_MSK,1 SignConfig=C:\myConfig\cmds.txt</code></p>
----------------	---

<b>Output</b>	Upon successful execution of the command, returns the internal configuration of the LAN device.				
	dev	name	prot	port	address
	-----				
	1	PCI:/dev/cs2a	TCP	288	10.17.1.9
	1	PCI:/dev/cs2a	TCP	288	192.168.5.203
	1	PCI:/dev/cs2a	TC6	288	fe80::0203:2dff:fe1c:6e8c

On a LAN device, the included PCIe card is accessible by setting up several parameters:

- `name`  
either a local PCIe card or a second LAN device (TCP)
- `prot`  
used protocol; either TCP or UDP (UDP has a packet size limitation)

- `port`  
port the TCP-Server (daemon) listens to (default: 288, each device can be set up with multiple ports)
- `address`  
interface address (either the TCP/IP address of the LAN device, the localhost address 127.0.0.1 or the Multicast address `224.1.0.1`)

### 4.10.3 CSLGetVersion

This command displays the version number of the LAN device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLGetVersion</code>
<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>Example</b>	<code>csadm Dev=10.17.1.9 CSLGetVersion</code>
<b>Output</b>	Upon successful execution of the command, the version number of the LAN device is returned. <code>CSLAN 4.1.0</code>

### 4.10.4 CSLGetStatus

The `CSLGetStatus` command displays status information about the LAN device.



This command is only implemented for LAN device with version 3.3.0 and later. For older versions of the LAN device, the output of this status information is not supported.  
Most of the information is only available for CryptoServer LAN V4 and later.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLGetStatus</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=10.17.1.9 CSLGetStatus</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, the status information of the LAN device is returned.
---------------	---

<b>Parameter</b>	<b>Description</b>
uptime	System uptime (days, hours)
fan speed [rpm]	Fan speed for all fans as rotation per minute (rpm). For a CSLAN V4b, three values are shown. The first value is for the CPU fan, the second is for the first internal fan, and the third is for the second internal fan. For a CSLAN V4c, two values are shown. The first value is for the first internal fan, and the second is for the second internal fan. This status information is only available for CryptoServer LAN V4 and later. A value of 0 for the fan speed indicates a broken fan. In this case, create an RMA (Return Merchandise Authorization) according to the Chapter "Contact Address for Support Queries".
CPU temperature [C]	CPU temperature for CPU of the CryptoServer LAN motherboard in degree Celsius. This status information is only available for CryptoServer LAN V4 and later.
redundant power supply	<ul style="list-style-type: none"> <li>Status of the redundant power supply units which can be OK or FAILED.</li> </ul> This status information is only available for CryptoServer LAN V4 and later

Table 25: Meaning of the output parameters of csadm CSLGetStatus

## 4.10.5 CSLGetLogFile

This command displays the log file of the LAN device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLGetLogFile[=&lt;fileno&gt;]</code>
---------------	--

<b>Parameter</b>	<b>Description</b>
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;fileno&gt;</code>	Number of the logfile on the LAN device that shall be from 0 to 9. If omitted, the first logfile (number 0, <code>/var/log/csxlan.log</code> ) is retrieved. If the number is invalid, the error code 0xB9067005 is returned

<b>Example</b>	<code>csadm Dev=10.17.1.9 CSLGetLogFile</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, the content of the log file is returned.
---------------	--



Use '>' to dump the output into a file, for example,  
`csadm GetConfigFile > c:\temp\csxlan.conf`

The log file is formatted according to the UNIX style.



The content of the log file depends on the trace level setting of the LAN device. See the respective Administration Manual for details.

#### 4.10.6 CSLGetConfigFile

This command displays the content of the configuration file `/etc/csxlan.conf` of the LAN device.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLGetConfigFile</code>
---------------	--



Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	csadm Dev=10.17.1.9 CSLGetConfigFile
----------------	--------------------------------------

<b>Output</b>	Upon successful execution of the command, the content of the configuration file is returned.
---------------	--



Use '>' to dump the output into a file, for example,  
`csadm GetConfigFile > c:\temp\csxlan.conf`

The configuration file is formatted according to the UNIX style.

## 4.10.7 CSLPutConfigFile

This command imports a new configuration file into a LAN device.

Independently from the given file name the configuration file is imported as `/etc/csxlan.conf`.

The imported file may also be formatted according to the operating system style. On the LAN device the file is reformatted automatically to the UNIX format style.

<b>Syntax</b>	csadm [Dev=<device>] CSLPutConfigFile
---------------	---------------------------------------

<b>Authentication</b>	Root password of the LAN device, see parameter <password>
-----------------------	---

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<password>	Root password of the LAN device <ul style="list-style-type: none"> <li>for hidden password entry use the string ask We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

<b>Parameter</b>	<b>Description</b>
<file>	Configuration file to be imported

<b>Example</b>	<code>csadm Dev=10.17.1.9 CSLPutConfigFile rootpassword configfile</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is returned.
---------------	--

### 4.10.8 CSLSetTracelevel

This command sets the trace level (log level) of a LAN device. Depending on the trace level, more or less information is written into the log file `/var/log/csxlan.log`.



The new trace level is only a temporary setting. The next time LAN device is (re)started, the level specified by the `LogLevel` variable (CSLANOS version  $\geq 5.0.0$ ) in the `/etc/csxlan.conf` configuration file is used again.

The following table shows the values that are supported for CSLANOS version  $\geq 5.0.0$ .

<b>Value</b>	<b>Trace level</b>	<b>Description</b>
0x03	Error	Error messages. The values 0x00, 0x01 and 0x02 have the same effect.
0x04	Warning	Warning messages
0x05	Notice	Normal but significant condition
0x06	Info	Informational messages
0x07	Debug	Debug level messages

Table 26: Trace levels for CSLANOS version  $\geq 5.0.0$

The `- 0x01` value is not supported by the `csadm CSLSetTracelevel` command although the corresponding `OFF` value of the `LogLevel` variable in the `/etc/csxlan.conf` file is supported.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLSetTraceLevel=&lt;password&gt;,&lt;level&gt;</code>
---------------	---

<b>Authentication</b>	Root password of the LAN device, see parameter <code>&lt;password&gt;</code>
-----------------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device <ul style="list-style-type: none"> <li>▪ for hidden password entry use the string ask</li> <li>▪ We strongly recommend using a hidden password entry.</li> <li>▪ root password of the LAN device in clear text</li> </ul>
<code>&lt;level&gt;</code>	Desired trace level as stated in the table above, for example, <code>0x06</code> .

<b>Example</b>	<code>csadm Dev=192.168.4.1 CSLSetTracelevel=123456,0x06</code>
----------------	---

<b>Output</b>	Upon successful execution of the command, no output is returned.
---------------	--

## 4.10.9 CSLShutdown

This command shuts down the LAN device as the Linux command `shutdown -h` would do at the local console. All programs and services are stopped, all devices are unmounted and the system is put into `RunLevel 0`. The LAN device can then be powered off.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLShutdown=&lt;password&gt;</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <code>&lt;password&gt;</code>
-----------------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device <ul style="list-style-type: none"> <li>▪ for hidden password entry use the string ask</li> <li>▪ We strongly recommend using a hidden password entry.</li> <li>▪ root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm Dev=192.168.4.1 CSLShutdown</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, no output is returned.
---------------	--

#### 4.10.10 CSLReboot

This command reboots the LAN device the same way the Linux command `shutdown -r` would do at the local console. All programs and services are stopped, all devices are unmounted and the system is rebooted again.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLReboot=&lt;password&gt;</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <code>&lt;password&gt;</code>
-----------------------	--

Parameter	Description
<code>&lt;device&gt;</code>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<code>&lt;password&gt;</code>	Root password of the LAN device <ul style="list-style-type: none"> <li>for hidden password entry use the string ask</li> <li>We strongly recommend using a hidden password entry.</li> <li>root password of the LAN device in clear text</li> </ul>

<b>Example</b>	<code>csadm Dev=192.168.4.1 CSLReboot</code>
----------------	--

<b>Output</b>	Upon successful execution of the command, the device reboots.
---------------	---

#### 4.10.11 CSLGetTime

This command reads the local system time and UTC time of the LAN device (not the clock of the PCIe card) and outputs the date and time.

To get the time of the PCIe card, perform the `csadm GetTime` command instead.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLGetTime</code>
---------------	--

Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.

<b>Example</b>	<code>csadm Dev=192.168.4.1 CSLGetTime</code>
----------------	---

<b>Output</b>	<p>Upon successful execution of the command, the date and time of the LAN device are returned.</p> <p>The first line shows the time of the LAN device in the time zone of the LAN device (local time).</p> <p>The second line shows the same time converted into the GMT (UTC) time zone (UTC time). This is not necessarily the time of the device (PCIe card).</p> <p>The date is shown in the DD.MM.YYYY format.</p> <pre>date: 03.03.2015    time: 10:58:50 (local time) date: 03.03.2015    time: 10:58:50 (UTC time)</pre>
---------------	--

### 4.10.12 CSLSetTime

This command sets the local system time of the LAN device (not the clock of the PCIe card).

To set the time on the PCIe card, perform the `csadm SetTime` command instead.



The LAN device retrieves its time from a time source. This time source might be an NTP server. As of CSLANOS 5.1, a PCIe clock card is supported as a time source as well.

If you set the time on the LAN device manually by performing the `csadm CSLSetTime` command and if this causes the time difference between the time on the LAN device and the time of the time source to be larger than 1000 s, this causes an error message and the NTP daemon is terminated automatically. In such case, the time on the device will not be set.

<b>Syntax</b>	<code>csadm [Dev=&lt;device&gt;] CSLSetTime=&lt;password&gt;,&lt;time&gt;</code>
---------------	--

<b>Authentication</b>	Root password of the LAN device, see parameter <password>
-----------------------	---

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<password>	Root password of the LAN device: <ul style="list-style-type: none"> <li>▪ For hidden password entry use the string ask</li> <li>▪ We strongly recommend using a hidden password entry.</li> <li>▪ root password of the LAN device in clear text</li> </ul>
<time>	<ul style="list-style-type: none"> <li>▪ YYYYMMDDhhmmss The time is set manually by using the format sample YYYYMMDDhhmmss YYYY=year, MM=month, DD=day, hh=hour, mm=minute, ss=second</li> <li>▪ SYSTEM The system time of the administration computer is used</li> </ul>

<b>Example</b>	csadm Dev=192.168.1.1 CSLSetTime=123456,SYSTEM
----------------	--

<b>Output</b>	<p>Upon successful execution of the command, the date and time of the LAN device are returned.</p> <p>The second line shows the time of the LAN device in the time zone of the LAN device (local time).</p> <p>The third line shows the same time converted into the GMT (UTC) time zone (UTC time). This is not necessarily the time of the device (PCIe card).</p> <p>The date is shown in the DD:MM.YYYY format.</p> <p>Time successfully set to:</p> <p>date: 06.12.2019    time: 11:16:53 (local time)</p> <p>date: 06.12.2019    time: 10:16:53 (UTC time)</p>
---------------	--

### 4.10.13 CSLGetSerial

This command reads and outputs the serial number of the LAN device appliance (not the serial number of the mounted PCIe card).

<b>Syntax</b>	csadm [Dev=<device>] CSLGetSerial
---------------	-----------------------------------

<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the environment variable CRYPTOSERVER.
<b>Example</b>	csadm Dev=192.168.1.1 CSLGetSerial
<b>Output</b>	Upon successful execution of the command, the serial number of the LAN device is returned.

#### 4.10.14 CSLGetLoad

This command reads and outputs the work load of the PCIe card in percent.

<b>Syntax</b>	csadm [Dev=<device>] CSLGetLoad
<b>Parameter</b>	<b>Description</b>
<device>	Device address This parameter can be omitted if the device address has been set as the CRYPTOSERVER environment variable. <device> = UDP:224.1.0.1 (Multicast) if the whole network shall be scanned.
<b>Example</b>	csadm Dev=192.168.0.1 GetLoad
<b>Output</b>	Upon successful execution of the command, the work load of the device (ratio of the time that requests/commands spend on the PCIe card to the total time) is returned.



The information of the work load of the device is read from the display module of the LAN device. If the display module is busy (i.e. an operator is working at the display) this information is not available and the command does not output any value.



- CryptoServer LAN (V4)  
The value returned by the command is not the work load at that time but is an average value of the last 60 seconds. It is recalculated and updated every 5 seconds.
- CryptoServer LAN V5  
The value returned by the command is not the work load at that time. It is an average value of the last 64 measurements. The value is continuously recalculated. If no packets arrive, it is recalculated once per second. If more packets arrive, it is recalculated more often, up to once per every few milliseconds.

#### 4.10.15 CSLListPPApps

Some applications (firmware modules) require a PIN pad directly connected to a USB port of the device or to another computer, for example, for a secure input of a Master Backup Key. To easily execute such a command on a LAN device, the corresponding function of the firmware module can be registered as a PIN pad application. Now the command is listed in the PIN Pad Applications menu of the LAN device.

`CSLListPPApps` shows all commands that have been registered in this way.



PIN pad applications are intended to be used on a LAN device without the need of connecting a monitor and keyboard to it. A specific PIN pad application can be selected via the LAN device menu as a submenu item of the PIN Pad Applications menu item. For further user guidance, the instructions on the PIN pad display are to be followed.

Alternatively, PIN pad applications can be executed like any other command using the function code FC (module ID) and subfunction code SFC (command ID) returned by the command `CSLListPPApps`. Watch the PIN pad display for further user guidance.

#### Syntax

```
csadm [Dev=<device>] CSLListPPApps
```



Parameter	Description
<device>	Device address This parameter can be omitted if the device address has been set as the CRYPTOSERVER environment variable. <device> = UDP:224.1.0.1 (Multicast) if the whole network shall be scanned.

<b>Example</b>	csadm Dev=10.17.1.9 CSLListPPApps
----------------	-----------------------------------

<b>Output</b>	Upon successful execution of the command, the work load of the device (ratio of the time that requests/commands spend on the PCIe card to the total time) is returned.
---------------	--

<b>Syntax</b>	csadm [Dev=<device>] CSLListPPApps
<b>Parameter</b>	<device> Device address of the LAN device This parameter can be omitted if the device address has been set as the CRYPTOSERVER environment variable.
<b>Authentication</b>	None
<b>Example</b>	csadm Dev=10.17.1.9 CSLListPPApps
<b>Output</b>	Upon successful execution of the command, the relevant commands are listed: 1. Import AES MBK from smartcard FC=0x69 SFC=2 Data= 2. Generate AES MBK on smartcard FC=0x69 SFC=4 Data= 3. List MBKs on smartcard FC=0x69 SFC=5 Data= 4. Copy MBK smartcard FC=0x69 SFC=6 Data= 5. Change MBK smartcard PIN FC=0x69 SFC=7 Data= 6. Import MBK from PIN pad & write to SC FC=0x69 SFC=8 Data= 7. Generate AES Key Shares & store on SC FC=0x69 SFC=10 Data=



The registration of a PIN pad application shall be implemented during the development of the firmware module.

You can start a PIN pad application by using the csadm command `csadm Cmd=FC,SFC,Data`.

FC, SFC and Data are the parameters included in the list of PIN pad applications.

## 5 Basic 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 (permission level 2 in user groups 6 and 7). All user(s) who want to perform the administrative tasks should have their authentication token at hand.

For the first set-up of the device in particular the Default Administrator Key of default administrator user ADMIN is needed.

### 5.1 Setting Up a New CryptoServer

After the device shipment the customer has to perform some initial steps to set up the device system, mainly:

- Change the authentication token of the default user ADMIN
- Create other users
- Generate a Master Backup Key (MBK)
- Import the MBK into the device

#### Preconditions

- The (LAN) device and the csadm tool are installed and running.
- The user authentication key of the default user ADMIN is at hand (either as keyfile key provided within the product bundle, or on the delivered smartcards with default PIN 123456).
- For every new user who shall be created on the device, the previously generated individual authentication token has to be at hand.
- As many smartcards as required for the MBK generation and import (according to the company security policy) have to be at hand.

#### Procedure

1. Execute `csadm GetState.`

Example:

```
csadm Dev=111.112.1.13 GetState
```

The device should be in Operational or in Maintenance Mode.

Alarm should be OFF.

Example:

```
mode = Operational Mode
state = INITIALIZED (0x00100004)
temp = 36.1 [C]
alarm = OFF
bl_ver = 5.00.5.5 (Model: Se-Series Gen2)
uid = 6e000018 850bbe01 | =*
adm1 = 53653530 20202020 43533434 34383739 | Se1500 CS6000024
adm2 = 53656375 72697479 53657276 65720000 | SecurityServer
adm3 = 494e5354 414c4c45 44000000 00000000 | INSTALLED
```

2. Replace the key of the default user ADMIN by the previously generated new authentication token. Use the csadm command `ChangeUser`, which has to be authenticated by the ADMIN himself.

Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 ChangeUser=ADMIN,:cs2:cjo:USB
```



You can skip this step, and the following one if your company's security policy requires authentication according to the two-person rule.

3. Follow the instructions on the PIN pad.  
First you are prompted to insert the smartcard where the default `ADMIN.key` is stored on into the PIN pad, and to enter the smartcard PIN. After that that you are prompted to insert the smartcard where the new authentication token is stored on, and to enter the smartcard PIN.



For the implementation of the two-person rule, instead of only replacing the user authentication key `ADMIN.key` for the user ADMIN by an individual one, the user ADMIN can be deleted after one or more user with sufficient permissions (minimum resulting permission 21000000) have been created.

To implement the two-person rule perform step 4 instead of step 2.

4. Create other users on the device by using the `csadm` command `AddUser`, possibly replacing the default user ADMIN.

Example:

Creating two users who can replace the default user ADMIN; one user for user management (two-person rule) and a cryptographic user

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0
AddUser=Admin1,11000000,rsasign,E:\myKeys\myRSA.key
AddUser=Admin2,10000000,rsasign,E:\myKeys\myRSA.key
AddUser=Admin3,11000000,rsasign,:cs2:cjo:USB0
AddUser=CryptoUser,20000001{CXI_GROUP=*},hmacpwd,ask
```

5. Follow the instructions on the PIN pad.



After you have successfully created at least two users with resulting minimum permission 21000000 you can optionally delete the default ADMIN user.

6. Set the time of the device clock with the `csadm` command `SetTime`.

Example:

```
csadm LogonSign=Admin1,:cs2:cjo:USB0 LogonSign=Admin3,:cs2:cjo:USB0
SetTime=GMT
```

7. Follow the instructions on the PIN pad display to authenticate the command `SetTime`.
8. Generate a Master Backup Key for the device by using the `csadm` command `MBKGenerateKey`. For the following example you have to keep three smartcards at hand for the MBK generation as well as the smartcards of both users who have to authenticate the command.

Example:

```
csadm Dev=PCI:0 LogonSign=Admin1,:cs2:cjo:USB0  
LogonSign=Admin3,:cs2:cjo:USB0  
Key=:cs2:cjo:USB0,1,:cs2:cjo:USB0,2,:cs2:cjo:USB0,3  
MBKGenerateKey=AES,32,3,2,cs2MBK
```

9. Follow the instructions on the PIN pad display to authenticate the command `MBKGenerateKey`, and then to generate the MBK.
10. Change the PIN of all smartcards whereon a share of the MBK is stored by using the `csadm` command `MBKPINChange`.  
Example: The three generated MBK shares are stored on three smartcards, so the following command shall be repeated for all three smartcards.

```
csadm MBKPINChange=:cs2:cjo:USB0
```

11. Follow the instructions on the PIN pad display.
12. Import the MBK into the device by using the `csadm` command `MBKImportKey`.  
Example:

```
csadm Dev=PCI:0 LogonSign=Admin1,:cs2:cjo:USB0  
LogonSign=Admin3,:cs2:cjo:USB0 Key=:cs2:cjo:USB0,1,:cs2:cjo:USB0,2  
MBKImportKey=3
```

13. Execute the `csadm` command `MBKListKeys` to make sure that the MBK has been successfully imported into the device.  
Example:

```
csadm Dev=PCI:0 LogonSign=ADMIN,:cs2:cjo:USB0 MBKListKeys
```



The new device has been set up successfully.

## 5.2 Entering and Leaving the 'Administration only' Mode

The device can switch temporarily between the normal Operational Mode and the restricted *Operational Mode – Administration-Only* without being restarted. In *Operational Mode – Administration-Only*, only functions needed for the device administration are available, and all cryptographic functions are blocked.

The device can switch between the two modes with the `csadm SetAdminMode` command. The operating mode set with this command is only relevant until the next time the device is restarted.

The device can be in *Operational mode – Administration only* which allows the use of only administrative functions. All cryptographic services are blocked. To switch a device from *Operational Mode – Administration-Only* to *Operational Mode* and activate the cryptographic services, perform these steps.

### Prerequisites

- The device is in state `INITIALIZED`, as well as in *Operational Mode – Administration Only*.
- The authentication token(s) of a user (or a group of users) with at least user permission 22000000 is available. If the device is set up for the first time, this means that the authentication key of the default administrator ADMIN is at hand, provided by the u.trust Anchor administrator.

### Procedure

1. Perform the `csadm GetState` command.
2. Perform the `csadm SetAdminMode` command.  
Example: `csadm Dev=3001@194.168.4.107 LogonSign=adminUsr,:cs2:cjo:USB0 SetAdminMode=0`
3. Perform the `csadm GetState` command again to verify the mode of the device.



The device has successfully been set back to *Operational Mode*.

## 5.3 Generating a User Authentication Key

For every user who shall authenticate themselves towards the device with an RSA or ECDSA signature, a user authentication key has to be generated. Basically, the private part of the user authentication key can either be stored in a keyfile, which optionally is password-protected, or on a smartcard. As the private part of the key cannot be read out from the smartcard, the usage of smartcards is more secure and therefore recommended.



Before delivery, the Default Administrator ADMIN is created on every device as the default administrator. The authentication token of this user, labelled "ADMIN," is shipped as clear text keyfile ( `ADMIN.key` ) and is initially stored on every smartcard that is shipped by Utimaco IS GmbH (with default PIN 123456).

The customer should either replace the authentication token of the ADMIN with a self-generated RSA key (see command `ChangeUser` ) or create other users with sufficient permissions on the device and then delete the Default Administrator ADMIN.

### Prerequisites

- If smartcards shall be used, at least one smartcard per user has to be at hand. If the smartcard already contains a key (RSA-Key or ECC-Key), this key will be overwritten.
- A PIN pad has to be connected to the USB port of the computer where csadm is installed or to another computer



We strongly recommend the generation of the new authentication token directly on a trustworthy USB flash drive, that the generated keyfile is copied on as many smartcards (provided by Utimaco) as required by the security policy of your company, and the USB flash drive and the smartcards are stored in a secure safe.

### Procedure

1. Create a new RSA or ECDSA key pair as an encrypted keyfile by using the csadm command `GenKey` .

```
csadm NewPassword=ask KeyType=RSA GenKey=E:\myKeys\myRSA.key,2048,Admin1
generating RSA key: E:\myKeys\myRSA.key, 2048 bits, owner: Admin1
Enter New Passphrase:
```

Repeat New Passphrase:



The generated keyfile can already be used to authenticate towards the device.  
If the key shall not be copied on a smartcard, the following steps can be skipped.

2. Store the key on a smartcard by using the command `SaveKey`.

Example:

```
csadm KeyType=RSA PrvKey=E:\myKeys\myRSA.key#ask SaveKey=:cs2:cjo:USB0  
Enter Passphrase:
```

3. Follow the instructions on the PIN pad.
4. Check that the key is stored on the smartcard by using the csadm command `GetCardInfo`.

Example:

```
csadm GetCardInfo=:cs2:cjo:USB0
```

5. Follow the instructions on the PIN pad display.
6. Change the PIN of the smartcard by using the csadm command `ChangePIN`.

Example:

```
csadm ChangePIN=:cs2:cjo:USB0
```

7. Follow the instructions on the PIN pad display.  
Depending on your security policy, the following step can be omitted, if steps 2 to 7 will be performed several times. The additional smartcards that will be created that way, have to be used as a backup for the generated key.
8. Create a backup of the private key on two smartcards with the command `BackupKey`.  
A backup smartcard is used for key storage only. It cannot be used for authentication towards the device. A backup smartcard contains only one XOR-half of a key, so two backup cards are necessary to regain the complete key. The key halves can be read out of the smartcard to generate new administration smartcards containing the stored key



at a later time with the command `SaveKey`.

Example:

```
csadm KeyType=RSA PrvKey=E:\myKeys\MyRSA.key#ask BackupKey=:cs2:cjo:USB0
```

9. Follow the instructions on the PIN pad.
10. Change the PIN of the backup smartcard by using the `csadm` command `ChangePIN`.

Example:

```
csadm ChangePIN=:cs2:cjo:USB0
```

11. Follow the instructions on the PIN pad display.
12. Store the keyfile at a protected place (for example, on a USB flash drive in a safe) or delete it (in case you made a backup copy on minimum two smartcards).



A user authentication key has been generated successfully.

The new user authentication key can now be assigned to every new device user on creation (see the `AddUser` command) who shall use RSA or ECDSA signature authentication. Furthermore, the key can be assigned to existing device users using RSA or ECDSA signature authentication with the `ChangeUser` command.

The PIN pad (if needed) has to be connected to the computer where the `csadm` tool is running (USB port) or to another computer.

If the authentication of the command `AddUser` requires a smartcard and the source of the new user's public key is a smartcard too, follow the instructions on the display of the PIN pad. You'll have to insert the smartcard for the command authentication (old key) first, and then the smartcard containing the user's new authentication key.

## 5.4 Checking the Battery State

You can check the state of the (LAN) device batteries with the `csadm` command `GetBattState`. The output of this command shows the state of the batteries:

<b><i>csadm Command</i></b>	<b><i>Output (examples)</i></b>	<b><i>Meaning</i></b>
csadm Dev=4000@112.111.1.13 GetBattState	Carrier Battery: ok (3.070 V)  External Battery: ok (3.553 V)	All connected batteries of a LAN device contain sufficient charge.
csadm Dev=4000@112.111.1.14 GetBattState	Carrier Battery: low (2.650 V)  External Battery: ok (3.553 V)	The carrier battery of a LAN device has low power and must be exchanged.
csadm Dev=4000@112.111.1.15 GetBattState	Carrier Battery: ok (3.070 V)  External Battery: low (3.100 V)	The external battery of a LAN device has low power and must be exchanged.
csadm Dev=/dev/cs2.0 GetBattState	Carrier Battery: ok (3.068 V)  External Battery: absence	No external battery connected to the local PCIe card. The carrier battery of the local PCIe card contains sufficient charge.

Table 27: Examples for the output of the GetBattState command

## 5.5 Performing a Complete Clear of the CryptoServer

You want to clear all data stored inside the device and reload the complete firmware. This may be useful, for example, in the following situations:

### Case 1:

You want to securely clear all secret data inside a device.

### Case 2 (Emergency-only):

You have lost your customer-individual administrator keys and want to regain an administrable device system.



Please be aware that in any case you will lose all your customer-specific data that are stored inside the device, in particular all stored keys.

In emergency case 2 you will have to set the device back to the factory default settings in which it usually is delivered.

Follow the instructions for case 2 only if this emergency case is given, i.e., if you have lost your customer-individual administrator keys.

If you follow the instructions for case 1, you will get a similar result as with the instructions for case 2, but with the exception that all users in the user database that use a public key as authentication token will remain (i.e. only users with password mechanisms will be erased).

## Procedure

### Case 1:

1. Perform the normal Clear command (`csadm Clear=INIT`). This command has to be authenticated by a user with administrative rights.



All secret data (e.g. databases) and all users with password authentication on the device are deleted. Users with signature authentication (for example, the default user ADMIN) remained because only the public key part of their user authentication key is stored on the device.

2. Restart the device (command `Restart`).
3. Load the firmware module package `*.mpkg` by using the `csadm` command `LoadPkg`.  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadPkg=E:  
\SecurityServer\Firmware\SecurityServer-Se2-Series\SecurityServer-Se2-  
Series-X.XX.X.X.mpkg
```

4. Follow the instructions on the PIN pad display to authenticate the command `LoadPkg`.
5. Execute the `ListFirmware` command.  
Example:

```
csadm Dev=PCI:0 ListFirmware
```

6. Check if all firmware modules have been successfully initialized (initialization level is shown as `INIT_OK`).
7. Verify the correct versions of the firmware modules. If some firmware modules have not been initialized yet, use the `GetBootLog` command to analyze the problem.
8. Optionally, load individual firmware modules (`*.mtc`, see command `LoadFile`).  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadFile=C:\myModules\myModule-0.9.9.9.mtc
```

9. Follow the instructions on the PIN pad display to authenticate the command `LoadFile`.
10. Restart the device to activate the newly loaded firmware modules.  
Example:

```
csadm Dev=PCI:0 Restart
```

11. This step is optional and only for customers who develop their own firmware modules: Load your customer-specific Alternative Module Signature Key with the command `LoadAltMdlSigKey`. This key is stored in parallel to Utimaco's Module Signature Key and does not replace it.  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 ... LoadAltMdlSigKey=MyMdlmSig.key
```

12. Follow the instructions on the PIN pad display.
13. This step is optional and only for customers who develop their own firmware modules: Load the private part of the Firmware Encryption Key (command `LoadFwDecKey`).  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadFwDecKey=MyFwDec.key
```

14. Follow the instructions on the PIN pad display.

## Case 2:

1. Execute an External Erase on the device.

2. Perform the `ClearToFactoryDefaults` command ( `csadm Clear=DEFAULT` ). This command does not have to be authenticated.
3. Restart the device (command `Restart`).
4. Reset the device alarm (command `ResetAlarm`).



The device is cleared into delivery state and the default administrator ADMIN is restored with his initial user authentication key ( `ADMIN.key` ).

5. Load the firmware module package `*.mpkg` by using the `csadm` command `LoadPkg` .  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0  
LoadPkg=E:\SecurityServer\Firmware\SecurityServer-Se2-Series\SecurityServer-  
Se2-Series-X.XX.X.X.mpkg
```

6. Follow the instructions on the PIN pad display to authenticate the command `LoadPkg` .
7. Execute the `ListFirmware` command.  
Example:

```
csadm Dev=PCI:0 ListFirmware
```

8. Check if all firmware modules have been successfully initialized (initialization level is shown as `INIT_OK` ).
9. Verify the correct versions of the firmware modules. If some firmware modules have not been initialized yet, use the `GetBootLog` command to analyze the problem.
10. Optionally, load individual firmware modules ( `*.mtc` , see command `LoadFile` ).  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 LoadFile=C:  
\myModules\myModule-0.9.9.9.mtc
```

11. Follow the instructions on the PIN pad display to authenticate the command `LoadFile` .
12. Restart the device to activate the newly loaded firmware modules.  
Example:

```
csadm Dev=PCI:0 Restart
```

13. This step is optional and only for customers who develop their own firmware modules: Load your customer-specific Alternative Module Signature Key with the command `LoadAltMdlSigKey`. This key is stored in parallel to Utimaco's Module Signature Key and does not replace it.

Example:

```
csadm LogonSign=ADMIN,:cs2:cjo:USB0 ... LoadAltMdlSigKey=MyMdlmSig.key
```

14. Follow the instructions on the PIN pad display.
15. This step is optional and only for customers who develop their own firmware modules: Load the private part of the Firmware Encryption Key (command `LoadFWDecKey`).  
Example:

```
csadm LogonSign=ADMIN,:cs2:cjoyb:USB0 LoadFwDecKey=MyFwDec.key
```

16. Follow the instructions on the PIN pad display.



The device has successfully been cleared.

## 5.6 Gathering Diagnostic Information

Perform the following `csadm` commands in a command prompt/shell and save the output in a text file to send it to the manufacturer Utimaco for problem analysis:

```
csadm [Dev=<device>] GetState
csadm [Dev=<device>] GetBootLog
csadm [Dev=<device>] GetAuditLog
csadm [Dev=<device>] GetInfo
csadm [Dev=<device>] GetBattState
csadm [Dev=<device>] ListFiles
csadm [Dev=<device>] ListFirmware
csadm [Dev=<device>] ListUser
```

```
csadm [Dev=<device>] GetTime  
csadm [Dev=<device>] <Authentication> MBKListKeys
```

For csadm versions < 2.5.3, the command

```
csadm [Dev=<device>] <Authentication> MBKListKeys
```

does not need any authentication. In this case, replace this command by the following one:

```
csadm [Dev=<device>] MBKListKeys
```

## 6 Advanced Administration Tasks

This chapter describes advanced administration functions of the device primarily addressing customers who want to extend the standard functionality of the device with self-developed firmware modules providing specific cryptographic functions and commands.

### 6.1 Creating Self-Signed/Encrypted Firmware Modules

Self-programmed firmware modules have to be signed with a customer-individual Alternative Module Signature Key. The public part of this key has to be loaded into the device (see command `LoadAltMdlSigKey`), before the firmware module can be loaded into the device.



Firmware modules created by Utimaco IS GmbH are always signed with Utimaco's Module Signature Key. As the public part of this key is stored by every device the operating software is always able to verify the signature on load of any firmware module that was created by Utimaco IS GmbH.

Optionally, firmware modules can be encrypted with the public part of a customer-specific RSA key (Firmware Encryption Key).



Firmware modules created by Utimaco IS GmbH are not encrypted, because they do not contain any secret.

Customers who want to encrypt their self-programmed firmware modules should keep in mind that good cryptography relies on the confidentiality of keys but not on the confidentiality of algorithms

#### Prerequisites

- The firmware module has been compiled and exists as executable file (COFF format, `*.out`, `*.dll` or `*.so`) or already within a module container (`*.mtc`).
- You have generated your own Module Signature Key, i.e., the Alternative Module Signature Key.
- If you want to encrypt your firmware module, you have generated your own Firmware Encryption Key.



## Procedure

1. If still necessary, unwrap the executable from the Module Transport Container (MTC) and the - signed - Module Manufacturer Container (MMC) with command `RemoveMTC`.

Example:

```
csadm RemoveMTC="C:\Program
Files\Utimaco\SecurityServer\Firmware\demo_md1.mtc"
```

Example output:

```
removing MTC ... OK
removing MMC ... OK
```

The resulting executable file `*.out` is stored in the same directory as the given `*.mtc` file.

This step can be omitted if the firmware module is provided as executable (`*.out`, `*.dll` or `*.so`).

2. Create the new (encrypted) container with the command `MakeMTC`.

If you want to create an encrypted and self-signed firmware module

Example:

```
csadm Model=c86 MMCSignKey="C:\Program
Files\Utimaco\keys\MyMd1Sign_enc.key"#ask FWEncKey="C:\Program
Files\Utimaco\keys\MyFwEnc.key" MakeMTC="C:\Program
Files\Utimaco\SecurityServer\Firmware\demo_md1.out"
```

Example output:

```
demo_md1.out:
building MMC ...
Enter Passphrase:
OK
building MTC ...
OK
```

If you want to create non-encrypted self-signed firmware module

Example:

```
csadm Model=c86 MMCSignKey="C:\Program
Files\Utimaco\keys\MyMdlSign_enc.key"#ask MakeMTC="C:\Program
Files\Utimaco\SecurityServer\Firmware\demo.out"
Example output:
demo_mdl.out:
building MMC ...
Enter Passphrase:
OK
building MTC ...
OK
```

The resulting firmware module file `*.mtc` is stored in the same directory as the given executable `*.out` (`*.dll` or `*.so`) file.



The self-signed/encrypted firmware modules have successfully been created.

## 6.2 Loading a Self-Signed Encrypted Firmware Module into the CryptoServer

This chapter guides you through the steps which are necessary for loading your own self-signed and optionally encrypted firmware module into the device.

### Preconditions

- The self-generated Alternative Module Signature Key is at hand.
- If you want to use encrypted firmware modules, your self-generated Firmware Encryption Key is at hand.

### Procedure

1. Load the public part of the Alternative Module Signature Key into the device with the `csadm` command `LoadAltMdlSigKey`.

Example:

```
csadm Dev=PCI:0 LogonSign=SysADM,"C:\Program
Files\Utimaco\keys\sysadm.key"#ask LoadAltMdlSigKey="C:\Program
Files\Utimaco\keys\MdlSignRSA.key"
```

- If you want to use encrypted firmware modules, load the private part of the Firmware Decryption Key into the device by using the csadm command `LoadFWDecKey`.  
Example:

```
csadm Dev=PCI:0 LogonSign=SysADM,"C:\Program
Files\Utimaco\keys\sysadm.key"#ask LoadFWDecKey="C:\Program
Files\Utimaco\keys\FwencRSA.key"
```

- Load the (encrypted) firmware module into the device with the csadm command `LoadFile`.  
Example:

```
csadm Dev=PCI:0 LogonSign=SysADM,"C:\Program
Files\Utimaco\keys\sysadm.key"#ask LoadFile="C:\Program
Files\Utimaco\SecurityServer\Firmware\demo.mtc"
```

- Make sure that the firmware module has been successfully loaded on the device by using the csadm command `ListFiles`. The firmware module is listed as `<mdlname>.msc`.  
Example:

```
csadm Dev=PCI:0 ListFiles
```

Example output:

file name	size	module:	ID	type	version
-----					
--					
FLASH\CXIKEY.db	64646	-			
FLASH\VMbk1.db	97	-			
FLASH\adm.msc	43692	ADM	0x087	SDK	3.0.32.0
Administration Module SDK					
FLASH\aes.msc	43692	AES	0x08b	SDK	1.4.2.0
AES Module SDK					
FLASH\asn1.msc	15532	ASN1	0x091	SDK	1.0.3.6
Asn1 Module SDK					
FLASH\audit000000.log	2190	-			
FLASH\auditkey.db	1785	-			

FLASH\authkey.db	1785	-			
FLASH\cmds.msc	53932	CMDS	0x083	SDK	3.6.2.0
Command Scheduler (SDK)					
FLASH\cxi.msc	172204	CXI	0x068	SDK	2.3.1.0
Crypto eXtended Interf.SDK					
FLASH\db.msc	23212	DB	0x088	SDK	1.3.2.4
Database module (SDK)					
FLASH\demo.msc	32940	DEMO	0x100	C64	1.0.0.0
FLASH\dsa.msc	28332	DSA	0x08d	SDK	1.2.3.3
DSA Module SDK					
FLASH\eca.msc	67756	ECA	0x08f	SDK	1.1.14.0
Elliptic Curve Arith. (SDK)					
FLASH\ecdsa.msc	42156	ECDSA	0x09c	SDK	1.1.21.0
ECDSA Module SDK					
FLASH\exmp.db	20	-			
FLASH\exmp.msc	19116	EXMP	0x100	SDK	2.1.0.0
Example Module SDK					
FLASH\hash.msc	38060	HASH	0x089	SDK	1.0.12.3
Hash Module SDK					
FLASH\lna.msc	39084	LNA	0x08e	SDK	1.2.4.7
Long Number Arithmetic (SDK)					
FLASH\mbk.msc	34476	MBK	0x096	SDK	2.3.0.0
Master Backup Key Module SDK					
FLASH\mdlsigalt.key	267	-B9CB966B7A-			
FLASH\post.msc	32428	POST	0x004	SDK	1.0.0.2
POST Module SDK					
FLASH\pp.msc	34476	PP	0x082	SDK	1.3.2.0
PIN pad Driver (SDK)					
FLASH\sc.msc	21676	SC	0x085	SDK	1.2.0.4
Smartcard module (SDK)					
FLASH\smos.msc	113836	SMOS	0x000	SDK	5.6.1.0
Operating System SDK					
FLASH\swap.com	0	-			
FLASH\user.db	4752	-			
FLASH\util.msc	13996	UTIL	0x086	SDK	3.0.5.3
Utility Module SDK					
FLASH\vdes.msc	23212	VDES	0x081	SDK	1.0.10.0
DES Module SDK					
FLASH\vdex.msc	15532	VDX	0x0f5	SDK	1.0.0.2
Vendor Defined Example					
FLASH\vrdsa.msc	47276	VRSA	0x084	SDK	1.3.6.7
RSA Module SDK					
31 files 1007778 bytes, 65304064 bytes available					

- Restart the device with csadm [Restart](#) to start using the newly loaded firmware module.

Example:

```
csadm Dev=PCI:0 Restart
```

6. Make sure that the firmware module has been successfully initialized and can be used by the device. Use the csadm command `ListFirmware`.

Example:

```
csadm Dev=PCI:0 ListFirmware
```

Example output:

ID	name	type	version	initialization level
0	SMOS	SDK	5.6.1.0	INIT_OK
4	POST	SDK	1.0.0.2	INIT_OK
68	CXI	SDK	2.3.1.0	INIT_OK
81	VDES	SDK	1.0.10.0	INIT_OK
82	PP	SDK	1.3.2.0	INIT_OK
83	CMDS	SDK	3.6.2.0	INIT_OK
84	VRSA	SDK	1.3.6.7	INIT_OK
85	SC	SDK	1.2.0.4	INIT_OK
86	UTIL	SDK	3.0.5.3	INIT_OK
87	ADM	SDK	3.0.32.0	INIT_OK
88	DB	SDK	1.3.2.4	INIT_OK
89	HASH	SDK	1.0.12.3	INIT_OK
8b	AES	SDK	1.4.2.0	INIT_OK
8d	DSA	SDK	1.2.3.3	INIT_OK
8e	LNA	SDK	1.2.4.7	INIT_OK
8f	ECA	SDK	1.1.14.0	INIT_OK
91	ASN1	SDK	1.0.3.6	INIT_OK
96	MBK	SDK	2.3.0.0	INIT_OK
9c	ECDSA	SDK	1.1.21.0	INIT_OK
f5	VDX	SDK	1.0.0.2	INIT_OK
100	DEMO	C64	1.0.0.0	INIT_OK



Alternatively, you can first load the new encrypted firmware module into the device, and afterwards load the private part of the Firmware Encryption Key into the device. In this case the encrypted firmware module is not decrypted during loading, but internally marked as an `*.emc` file which will be decrypted and converted into an `*.msc` file as soon as the private part of the Firmware Encryption Key is loaded into the device with the csadm command `LoadFWDeckKey ..`



The self-signed encrypted firmware module has been loaded successfully.

## 6.3 Creating and Using the Signed Configuration File `cmds.scf`

You can use the signed configuration file `cmds.scf` to disable some specific external firmware interfaces or/and to set specific permission requirements (higher than the permissions required by default) for some external device functions.

This chapter shows you by means of examples how to create and use the file `cmds.scf`.

### Preconditions:

- The required min. versions of the firmware modules ADM, CMDS and SMOS are loaded on the device.
- You have created a configuration file `cfg` (or any other file extension) with the correct syntax. The syntax of the signed configuration file follows the syntax of configuration files.

Example:

If the configuration file should be used for hardening the device interface, and for configuring role-based access to selected device functions, it has to contain exactly one section `[DisableSFC]` and one section `[Permissions]` in the following format:

```
[DisableSFC]
0x083 = 0,1 # disable Echo and Reverse Echo in CMDS module
0x068 = 0   # disable VerifyGenuineness in CXI module

[Permissions]
0x083 = 12:FF000000,\ # CMDS module, increased authentication
                        # requirements for BackupUser
        6:FF000000,\ # CMDS module, increased authentication
                        # requirements for ChangeUser
        5:FF000000   # CMDS module, increased authentication
                        # requirements for DeleteUser
0x068 = 17:6F000000  # CXI module, increased authentication
                        # requirements for ImportKey
0x087 = 10:22000000  # ADM module,
                        # authentication required for GetAuditLog
```

- The version of the csadm tool installed on your administration computer is at least 1.8.11b.
- You have generated your own Alternative Module Signature Key
- You have loaded the public part of your Alternative Module Signature Key into the device with the csadm command `LoadAltMdlSigKey`

## Procedure

1. Sign the configuration file with the private part of the Alternative Module Signature Key.  
Example:

```
csadm MdlSignKey=D:\keys\myMDLSign.key#password SignConfig=N:\User\Utimaco\cmds.cfg
```

The signed configuration file is created and stored as file `cmds.scf` in the same directory as the unsigned configuration file version.

2. Load the signed configuration file into the device.  
Example:

```
csadm Dev=PCI:0 LogonSign=ADMIN,"C:\Program Files\Utimaco\SecurityServer\Administration\ADMIN.key"#ask LoadFile= N:\User\Utimaco\cmds.scf
```

3. Check that the signed configuration file `cmds.scf` has been successfully loaded into the device.  
Example:

```
csadm Dev=PCI:0 ListFiles
```

4. Restart the device.  
Example:

```
csadm Dev=PCI:0 Restart
```

5. Check the corresponding log file entries to ensure the signed configuration file has been loaded and the provided settings are used.

- a. Check the boot log file to ensure the settings configured in section [DisableSFC] are used:

Example:

```
csadm Dev=PCI:0 GetBootLog
```

You should see an entry for the functions that have been disabled.

Example:

```
CMDS/DSOM: 0x083 - disabled 0 1
```

Check that the functions you wanted to be disabled are blocked:

Example:

```
csadm Dev=PCI:0 cmd=0x083,0,1
```

You should see the following error message.

```
Error B0830061
CryptoServer module CMDS, Command scheduler
This function is not available in this HSM configuration
```

- b. Check the audit log file to ensure the settings configured in section [Permissions] are used:

Example:

By default, the csadm command GetAuditLog does not have to be authenticated.

```
csadm Dev=PCI:0 GetAuditLog
```

Since the command now has to be authenticated with min. permission 00200000 the device now returns an error message:



```
Error B0830001
CryptoServer module CMDS, Command scheduler
permission denied
```

With the appropriate authentication you should see the following audit log entries for the `cmds.cfg` used as example in the preconditions:

```
csadm dev=PCI:0 LogonPass=demo,ask GetAuditLog
19.02.16 14:10:43 SMOS Ver. 3.3.4.1 successfully started
19.02.16 14:10:44 FC:0x083 SFC:5 Configured Permission=FF000000
19.02.16 14:10:44 FC:0x083 SFC:6 Configured Permission=FF000000
19.02.16 14:10:44 FC:0x083 SFC:12 Configured Permission=FF000000
19.02.16 14:10:44 FC:0x087 SFC:10 Configured Permission=00200000
19.02.16 14:10:44 FC:0x068 SFC:17 Configured Permission=6F000000
```



The signed configuration file has been edited successfully.

## 6.4 Generating and Verifying Signed Audit Log Files

Proceed as follows to generate and verify signed audit log files.

1. When generating and using a signed audit log, an audit log signature key is required. For details, see *Audit Log Signature Key* in the [CryptoServer - Administration Manual \(p. 243\)](#). This key can be generated or retrieved only if the device is in Operational Mode, Therefore, first verify whether the device is in this mode by performing the `csadm GetState` command according to the following example.

```
csadm Dev=192.168.4.1 GetState | grep mode
```

The output should be as follows:

```
mode = Operational Mode
```

- The audit log signature key is stored in the `auditkey.db` database. This database contains only one audit log signature key and nothing else. Verify whether this database exists on your device by performing a command according to the following example:

```
csadm Dev=192.168.4.1 ListFiles | grep auditkey.db
```

If the output of this command starts with `FLASH\auditkey.db`, this database exists. Example output:

```
FLASH\auditkey.db 1785 -
```

If the `auditkey.db` database is not available, continue with step 3, otherwise, continue with step 4.

- Generate an audit log signature key by performing the `csadm GenerateAuditLogKey` command according to the following example.

```
csadm Dev=192.168.4.1 LogonSign=ADMIN,:cs2:cjo:USB0 GenerateAuditLogKey=1  
> ./pub_audit_log_key.txt
```

It is important that the output of the `csadm GenerateAuditLogKey` command is redirected into a file as shown in the example because this file is used later on to generate signed audit log files. For details, see [GenerateAuditLogKey \(p. 75\)](#). Continue with step 5.

- Retrieve the audit log signature key by performing the `csadm GetAuditLogKey` command according to the following example.

```
csadm Dev=192.168.4.1 GetAuditLogKey > ./pub_audit_log_key.txt
```

It is important that the output of the `csadm GetAuditLogKey` command is redirected into a file as shown in the example because this file is used later on to generate signed audit log files. For details, see [GetAuditLogKey \(p. 77\)](#). Continue with step 5.

- Verify which unsigned audit log files exist on the device.

```
csadm Dev=192.168.4.1 ListFiles | grep .log
```

Example output:

```
FLASH\audit13AC97.log 155713 -
```

Consider that the unsigned audit log files are stored in the FLASH memory of the device and that short or long audit log filenames are used for them. In contrast, signed audit log files are always generated on the computer csadm is running on and extra long audit file names are used for these audit log files, see *Audit Log Files* in the [CryptoServer - Administration Manual](#) (p. 243).

6. To generate signed audit log files on the computer csadm is running on, perform the `csadm GetSignedAuditLog` command according to the following example. Assign the redirected output of the `csadm GenerateAuditLogKey` command or the `csadm GetAuditLogKey` command (for example, the `pub_audit_log_key.txt` file) to the `AuditPubKey` parameter.

```
csadm Dev=192.168.4.1 LogonSign=ADMIN,:cs2:cjo:USB0 AuditPubKey=./  
pub_audit_log_key.txt GetSignedAuditLog=./logs
```

This example generates signed audit log file(s) in the logs directory. For details, see [GetSignedAuditLog](#) (p. 78).

7. To verify a signed audit log file, perform a command according to the following example.

```
csadm Dev=192.168.4.1 AuditPubKey=./pub_audit_log_key.txt  
VerifySignedAuditLog=./logs/CS123456_0013AC97.log,print
```

8. If you want to delete the unsigned audit log files in the FLASH memory of the device that are related to the signed audit log files on the computer csadm is running on at a later date, perform the `csadm ClearAuditLogFiles` command according to the following example.

```
csadm Dev=192.168.4.1 LogonSign=ADMIN,:cs2:cjo:USB0 ClearAuditLogFiles=3a,./  
logs
```

## 6.5 Enabling Mutual Authentication



Mutual authentication including the authentication of the device to the host application has been first introduced with the SecurityServer/CryptoServer SDK 4.10. The feature is disabled by default and must be manually enabled on demand.

To use mutual authentication, at least version 3.5.3.x of the firmware module CMDS shall be loaded on the device and successfully initialized. Additionally, the csadm tool provided on the SecurityServer/CryptoServer SDK 4.10 product CD must be used for the device administration.

To enable the device to prove its authenticity at the beginning of a Secure Messaging session, the following steps are required:

1. Execute the `csadm GetState` command to ensure that the device is in Operational Mode and no alarm is present.

Example:

```
mode      = Operational Mode
state     = INITIALIZED (0x00100004)
temp      = 31.9 [C]
alarm     = OFF
bl_ver    = 4.00.3.0           (Model: CSe-Series)
hw_ver    = 4.00.3.0
uid       = dd000016 be1eac01
adm1      = 43536531 30302020 43533530 35313536 | CSe100 CS505156
adm2      = 53656375 72697479 53657276 65720000 | SecurityServer
adm3      = 494e5354 414c4c45 44000000 00000000 | INSTALLED
```

2. Execute the `csadm GetHSMAuthKey` command to retrieve the public part of the HSM Authentication Key.

Example:

```
CS505156=AC5D8198246C4FF407F7E11B4E53C4AFF326BDF350FF531CB26E0A250DA7CD5C0A
E06C69CA8846D75102B80C3D59E06C6CC98130D27EA260CB0836A486EBE5A29A288EDFFCA43
07A6F3A9EF0549D343FDAB57012502050BC40AF8A71F41914402DE917427D976DB719802BFE
CA96FCCABEF32A7EF79296A4B5C79D30FC545A6C7ACE585B48890E243BC9A664E58E9992969
DA1A234581083A18C62DB9FEB3B05BB29E4E68C282220DF3939120F125AEC9F6D9533C8A9BC
3C890433B218444C81A1379A671DEDF92273E61282B345E23ECE82AF964E96BD4681678A590
77C1CE5F17A58D620D72FE845FB68D249BA7E25D924A98269E050D57F4558A29B6BCDE07C30
D6914BF6A06DF5FE9014C8AF3BE1040A96B36D3583976D180520DAF994E3A0A9C9D5DEE836F
1ACD606A3B8FAD186ED6AC0FC34C974FCCB5194F87EB3971F8A8DF728B02ECC2F21ECA4BCB7
```

```
1748435540EF92EDE0A69CA74E070D38857661D63538C64C4169F6EEE972D0B455734F40435B
7B927BFF8982F5AD3C998D304D
```

Additionally, the command output contains the device serial number (in format `CS<xxxxxx>`) that is the same as the one included in the `adm1` line of the `GetState` command output.

Example:

`GetHSMAuthKey` output:

```
CS505156      =AC5D8.....
```

`GetState` output:

```
adm1          = 43536531 30302020 43533530 35313536 | CSe100 CS505156
```

In case of the device Simulator, the serial number has the format `SI<xxxxxx>`. `SI` stands for “simulator” and `<xxxxxx>` stands for the port number the simulator is listening on, i.e., 003001 for the first simulator instance, 003003 for the second simulator instance etc.

Example: `SI003001`

In case of the PaymentServer Simulator, the serial number is always `CS000000`.

- Copy the complete and unchanged output of the `csadm GetHSMAuthKey` command to an empty text file.  
This file may contain multiple entries separated by a blank line, for example, if a device cluster is deployed.
- Save the text file, for example, under the file name `key`, at a well-chosen location on the computer where the host application is installed.



You can use `>` to redirect the output of the `csadm GetHSMAuthKey` into the text file.

Example:

```
csadm Dev=3001@127.0.0.1 GetHSMAuthKey > C:\ProgramData\Utimaco\HSMAuth.key
```

If the text file exists already, it is overwritten

5. Distribute the file containing the public part of the HSM Authentication Key, for example `HSMauth.key`, to all administration computers/persons in your organization that want to communicate with the device in a mutually authenticated way. Choose a trustworthy and authentic procedure for the file distribution.
6. Set the permissions for the file, so that only predefined system administrators are permitted to maintain the file content, but make sure that all users of HSM tools and applications have read access.
  - On a Linux computer, use the `chmod` command.
  - On a Windows computer, right-click the file, select **Properties** and then click the **Security** tab. If necessary, consult the appropriate Microsoft help for further details.
7. Create the system environment variable `CS_AUTH_KEYS` to contain the path to the previously created text file (for example, `HSMauth.key`) on all corresponding hosts, this all computers where the previously created text file has been distributed to.

Example:

- For Windows:  
Either you open **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment Variables > System variables > New**, enter `CS_AUTH_KEYS` in the Variable name field, enter, for example, `C:\ProgramData\Utimaco\HSMauth.key` in the Variable value field, and click **OK**,  
or  
you open **Start > All Programs > Accessories > Command Prompt > Run as administrator**, and perform the following command:

```
setx CS_AUTH_KEYS=C:\ProgramData\Utimaco\HSMauth.key /m
```

This setting is effective for all command-lines that will be opened in the future. If you want make it effective in the current command-line, perform the following command:

```
set CS_AUTH_KEYS=C:\ProgramData\Utimaco\HSMauth.key
```

If you want to remove the `CS_AUTH_KEYS` environment variable at a later date, either open **Start > Control Panel > System and Security > System > Advanced system settings > Advanced > Environment Variables > System variables**, select `CS_AUTH_KEYS`, and click **Delete**,

or

open **Start > All Programs > Accessories > Command Prompt > Run as administrator**, and perform the following command:

```
reg delete "HKLM\SYSTEM\CurrentControlSet\Control\Session  
Manager\Environment" /F /V CS_AUTH_KEYS
```

For Linux:

Make sure that you are in the home directory of a user, for example, `/home/user01`. Perform the following command:



It is very important that you use `>>` in the following command. It appends the preceding text to the specified file. Do not use `>`, because it overwrites the entire file.

```
echo export CS_AUTH_KEYS=~/.HSMauth.key >> .bashrc
```

This setting is effective for all command-lines that will be opened in the future. If you want make it effective in the current command-line, perform the following command:

```
. .bashrc
```

If you want to verify the current value in a command-line, perform the following command:

```
echo $CS_AUTH_KEYS
```

If you want to remove the `CS_AUTH_KEYS` environment variable at a later date, remove the following line from the `/home/user01/.bashrc` file

```
export CS_AUTH_KEYS=~/.HSMauth.key
```

and perform the following command to remove `CS_AUTH_KEYS` in the current command-line:

```
unset CS_AUTH_KEYS
```

In other Linux shells, the commands might be different.



Mutual Authentication has been enabled successfully.

---

From now on, every Secure Messaging session established between the device and any host application is mutually authenticated.

In case the content of the file containing the public part of the HSM Authentication Key has been modified, that is the HSMAuthKey verification failed or the file could not be found at the location specified by the environment variable `CS_AUTH_KEYS`, the Secure Messaging session fails to be established with an appropriate error message.

An error message is also returned if all sensitive data has been erased from the device as a consequence of an alarm or the execution of the Clear command, or if the device has been exchanged or removed. This way, any changes to the device will be noticed.

To set up mutual authentication again the device shall be inspected and, if trusted, the procedure described above shall be repeated.



## 7 Troubleshooting

This chapter provides solutions to some possible trouble cases when using the device. See the [CryptoServer - Troubleshooting \(p. 243\)](#) document for further troubleshooting details.

### 7.1 Checking the Operativeness and the State of the CryptoServer

This section deals with the situation where it is not known whether the device works at all, and in which mode/state it is. The following steps should be systematically performed to check device operativeness and, if possible, to get the device working again.

#### Prerequisites

- If the PCIe card is mounted in your local computer, make sure that the PCIe driver is running and the administration tool csadm is installed on the computer.
- If the is a part of a LAN device, make sure that the LAN device and the administration computer, whereon the administration tool csadm is installed, are properly connected to the network (try to 'ping' the LAN device from the administration computer).

#### Procedure

1. Perform the `csadm GetState` command.

```
csadm Dev=PCI:0 GetState
```

Example Output:

```
mode = Operational Mode
state = INITIALIZED (0x00100004)
temp = 36.1 [C]
alarm = OFF
bl_ver = 5.00.5.5 (Model: Se-Series Gen2)
uid = 6e000018 850bbe01 | =*
adm1 = 53653530 20202020 43533434 34383739 | Se1500 CS6000024
adm2 = 53656375 72697479 53657276 65720000 | SecurityServer
adm3 = 494e5354 414c4c45 44000000 00000000 | INSTALLED
```

In the following table, the operativeness check is done by considering the output data for the mode, state and alarm of the `csadm GetState` command.

<b>Result</b>	<b>Explanation/Reason/Adjustment</b>
mode = Operational Mode state = INITIALIZED alarm = OFF	Everything is Ok, → Continue with step 2.
mode = Operational Mode Administration-Only state = INITIALIZED alarm = OFF	Everything is Ok, → Continue with step 2.
mode = Maintenance Mode state = INITIALIZED alarm = OFF	Only the backup set of system firmware modules ( *.sys ) has been started. No cryptographic services like CXI, JCE, etc., are available. → Restart the device (step 3). If the device is still in Maintenance Mode, load the wanted firmware module package ( *.mpkg , see command LoadPKG , see <a href="#">LoadPkg (p. 160)</a> ). → Restart the device (step 3).
mode = Bootloader Mode state = INITIALIZED alarm = ON	Device is in Bootloader Mode. → Restart the device (step 3) csadm Dev=PCI:0 restart
	An alarm has occurred (and is possibly physically still present) → See <a href="#">Alarm Treatment (p. 235)</a> .
If state is not INITIALIZED	The device is not correctly initialized or even defect. → Please get in contact with the Utimaco IS GmbH. Use the contact data given in <a href="#">Contact Address for Support Queries (p. 242)</a> .
If error: B9011xxx, B9015xxx, B9016xxx, B9017xxx or B9021xxx until B9024xxx	The PCIe card of the device does not react. → Try a restart (step 3).
other errors: B901xxxx or B902xxxx	No connection to the device/LAN device. Communication problem, wrong host or device name, problem with the network. → Check parameters. → Perform ping at LAN device. → Check state/configuration of the TCP daemon on the LAN device.

2. Perform the `csadm ListFirmware` command.

```
csadm Dev=PCI:0 ListFirmware
```

Result	Explanation/Reason/Adjustment
All necessary modules are listed and initialized (INIT_OK).	Everything is OK. Device is in Operational and ready to use.
Some necessary modules are missing in the given list.	Modules are not loaded into the device. → Check the presence of the modules with the <code>csadm ListFiles</code> command. Load the missing modules with the <code>csadm LoadFile</code> and restart the device. If the modules cannot be started: → Perform the <code>csadm GetBootLog</code> command and search the boot log for errors.
At least one module is not initialized (i.e., not INIT_OK or INIT_INACTIVE).	Firmware module(s) cannot be started (for example, module dependencies cannot be resolved). → Perform the <code>csadm GetBootLog</code> command and search the boot log for errors.

3. Perform the `Restart` command.

```
csadm Dev=PCI:0 Restart
```

Result	Explanation/Reason/Adjustment
no error other error	OK → go back to step 1 → Switch the device power off and on again, then go back to step 1 If this does not help: may be hardware problem → Please contact Utimaco IS GmbH. Use the contact data given in <a href="#">Contact Address for Support Queries (p. 242)</a> .

## 7.2 Alarm Treatment

An alarm can be triggered on the device for various physical reasons, like for example, the temperature being too high or too low, empty battery, or after an External Erase had been executed, see *An Alarm and Its Consequences* in the [CryptoServer - Administration Manual \(p. 243\)](#) for a list of all possible alarm reasons and a detailed description of the alarm mechanism of the device.

Most alarm reasons can be removed (for example, exchange low battery or cool down high temperature). The `GetState` command shows the reason for an alarm, and whether the alarm is still present, see [GetState \(p. 56\)](#). If the reason for an alarm cannot be removed (for

example, defect foil) then please get in contact with the manufacturer/Utimaco IS GmbH. Otherwise you can reset the pending alarm state.

### Prerequisites

An alarm has occurred to the device. This will be announced with the `csadm 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.

### Procedure

1. If the alarm is physically still present, remove the alarm cause if possible. Then restart the device with the `csadm Restart` command.
2. Execute the `csadm GetState` command again. Even if the reason for the alarm has been removed, the alarm state will still be `ON`, but the alarm should no longer be shown as present (only as has occurred).
3. Depending on the alarm state proceed as follows:  
 If the alarm is still shown as present, contact the customer support team of Utimaco. Use the contact data given in [Contact Address for Support Queries \(p. 242\)](#).  
 If the alarm is shown as has occurred, proceed as follows:
  - a. Perform the `csadm ResetAlarm` command
  - b. Restart the device ( `csadm Restart` ).
  - c. Execute `GetState` again. The alarm state should now be `OFF`. Under certain circumstances audit log file names `audit_<2-digit hexadecimal number>.log` instead of `audit<6-digit hexadecimal number>.log` are used in the Maintenance Mode. If this is the case and if the `csadm ResetAlarm` command has been performed successfully, the device enters the Operational Mode after the restart and all `audit_<2-digit hexadecimal number>.log` files are renamed to a `udit<6-digit hexadecimal number>.log` files, see *Audit Log File Names* in the [CryptoServer - Administration Manual \(p. 243\)](#).



Please be aware that all users who have used a password authentication mechanism are lost after an alarm. These users have to be replaced, if needed.

If self-developed encrypted firmware modules have been loaded and stored on the device, all they are deleted after an alarm. In this case you have to do the following steps:

4. Load the private part of Firmware Encryption Key into the device again (command `LoadFWDecKey` ).
5. Load all necessary (encrypted) firmware modules or your firmware module package again.

## 7.3 Leaving Error State

This chapter explains how to leave the error state. The error state is indicated in the output of the `csadm GetState` command by the following entry:

```
error state (<hexadecimal error code>), for example, error state  
(0xB0040079)
```

Depending on the error cause, this can require various activities.

If the device is in DEFECT state, which means that the `csadm GetState` command returns `state = DEFECT` , contact the customer support team of Utimaco. Use the contact data given in [Contact Address for Support Queries \(p. 242\)](#).

### Prerequisites

- The following users are available:
  - One or several users with the Administrator role (sum of minimum permissions 2 in the user group 6, 02000000)
  - One or several users with the User Administrator role (sum of minimum permissions 2 in the user group 7 (20000000))
  - Have the `user.db` , the `cxiskey.db` and, if available, the `auditkey.db` databases at hand that you previously should have backed up by using the `csadm ... BackupDatabase` command.
- All users have to keep their authentication tokens (smartcards or keyfiles) at hand.
- The smartcards the MBK shares are stored on are at hand.
- Keep the operating manuals at hand. You find them on the delivered product CD in the `\Documentation\Operating Manuals\<English|German>` directory.

### Procedure

1. As of SecurityServer 4.30 / CryptoServer SDK 4.30, the `csadm GetAuditLog` command is available in the error state. This might provide useful information to solve the situation.
2. Perform the `csadm Restart` command or power cycle the device.
3. Check with the `csadm GetState` command, if the device is still in error state.
  - a. If this is not the case, your device is ready-to-use.
  - b. Otherwise, proceed with step 4.
4. Perform the `csadm Clear` command according to the following example.

```
csadm Dev=PCI:0 LogonSign=CSAdmin1,:cs2:cjo:USB0 Clear=Init
```

5. Check with the `csadm GetState` command, if the device is still in error state or in Maintenance Mode and INITIALIZED state.
  - If the device is in Maintenance Mode and INITIALIZED state, continue with the steps in case 1 in [Performing a Complete Clear of the CryptoServer \(p. 210\)](#).
  - Otherwise, proceed with step 6.
6. Perform an External Erase.
  - If you are using a device card mounted in a host computer, perform an External Erase as described in case 2 in [Performing a Complete Clear of the CryptoServer \(p. 210\)](#).
  - If you are using a LAN V5 device, perform an External Erase, see *Deleting All Sensitive Data* in the [CryptoServer LAN V5 - Operating Manual \(p. 243\)](#).
7. Execute the `csadm GetState` command.

The device shall be now in Maintenance Mode and in alarm state (alarm = ON). The reason for the alarm is no longer present (`Alarm has occurred`).

Example output:

```
mode      = Maintenance Mode
state     = INITIALIZED (0x000a7f84)
temp      = 44.0 [C]
alarm     = ON
sens      = 027f
           - Alarm has occurred
           - external Erase is executed
```

```
bl_ver      = 5.01.0.5           (Model: Se-Series Gen2)
hw_ver      = 5.01.0.0
uid         = f2000018 850be801 |
adm1        = 53653135 30302020 43533630 30303332 | Se1500 CS600032
adm2        = 53656375 72697479 53657276 65722020 | CryptoServer
adm3        = 494e5354 414c4c45 44202020 20202020 | INSTALLED
```

If the device is still in error state, contact the customer support team of Utimaco IS GmbH. You find the contact data in [Contact Address for Support Queries \(p. 242\)](#).

8. Reset the alarm by using the `csadm ResetAlarm` command.

```
csadm Dev=PCI:0 LogonSign=CSAdmin1,:cs2:cjo:USB0 ResetAlarm
```

The CryptoServer is automatically restarted. Under certain circumstances that are described in *Audit Log File Names* in the [CryptoServer - Administration Manual \(p. 243\)](#), audit log file names `audit_<2-digit hexadecimal number>.log` instead of `audit<6-digit hexadecimal number>.log` are used in the Maintenance Mode. If this is the case and if the `csadm ResetAlarm` command has been performed successfully, the device enters the Operational Mode after the restart and all `audit_<2-digit hexadecimal number>.log` files are renamed to `audit<6-digit hexadecimal number>.log` files.

9. This step is optional. Only perform it, if the steps in , have been executed.

Execute the `csadm GetHSMAuthKey` command to force the generation of a new device authentication key and to retrieve its public part.

Example:

```
csadm Dev=PCI:0 GetHSMAuthKey
```

Example output:

```
CS600032=AC5D8198246C4FF407F7E11B4E53C4AFF326BDF350FF531CB26E0A250DA7CD5C0AE
06C69CA8846D75102B80C3D59E06C6CC98130D27EA260CB0836A486EBE5A29A28EDFFCA4307
A6F3A9EF0549D343FDAB57012502050BC40AF8A71F41914402DE917427D976DB719802BFECA9
6FCCABEF32A7EF79296A4B5C79D30FC545A6C7ACE585B48890E243BC9A664E58E9992969DA1A
234581083A18C62DB9FEB3B05BB29E4E68C282220DF3939120F125AEC9F6D9533C8A9BC3C890
433B218444C81A1379A671DEDF92273E61282B345E23ECE82AF964E96BD4681678A59077C1CE
5F17A58D620D72FE845FB68D249BA7E25D924A98269E050D57F4558A29B6BCDE07C30D6914BF
6A06DF5FE9014C8AF3BE1040A96B36D3583976D180520DAF994E3A0A9C9D5DEE836F1ACD606A
3B8FAD186ED6AC0FC34C974FCCB5194F87EB3971F8A8DF728B02ECC2F21ECA4BCB7174843554
0EF92EDE0A69CA74E070D38857661D63538C64C4169F6EEE972D0B455734F40435B7B927BFF8
982F5AD3C998D304D
```

The command output contains the serial number of the device in format `CS<XXXXXX>` that is the same as the one included in the `adm1` line of the `csadm GetState` command output.

For example:

```
GetHSMAuthKey output:
CS600032 =AC5D8.....
GetState output:
adm1 = 53653530 20202020 43533434 34383739 | Se1500 CS600032
```

In case of the device Simulator, the serial number has the format `SI<XXXXXX>`. `SI` stands for "simulator" and `<XXXXXX>` stands for the port number the simulator is listening on, i.e., 003001 for the first simulator instance, 003003 for the second simulator instance etc.

Example: `SI003001`

In case of the PaymentServer Simulator, the serial number is always `CS000000`.

10. Open the `HSMauth.key` file you previously have created in [Enabling Mutual Authentication \(p. 228\)](#)
11. Delete the currently available content.
12. Copy the complete and unchanged output of the `csadm GetHSMAuthKey` command to the `HSMauth.key` file.
13. Save the changes you made to the `HSMauth.key` file and close the file.
14. Load the firmware package `SecurityServer-Se2-Series-X.XX.X.X.mpkg` provided on the product CD in the `\Firmware\SecurityServer-Se2-Series` directory. Use the `csadm LoadPkg` command. It must be authenticated by at least two users with the Administrator role and minimum sum of permissions 2 in the user group 6 (02000000).

```
csadm Dev=PCI:0 LogonSign=ADMIN,:cs2:cjo:USB0
LogonSign=CSAdmin1,:cs2:cjo:USB0 LoadPkg=F:
\Firmware\SecurityServer-Se2-Series\SecurityServer-Se2-
Series-X.XX.X.X.mpkg
```

15. Check that all loaded firmware modules are successfully initialized and running with the `csadm ListFirmware` command.  
Syntax: `csadm [Dev=<device>] ListFirmware`



Example:

```
csadm ListFirmware
```

16. Set the time and date of the device.

Example with the `csadm SetTime` command:

```
csadm Dev=PCI:0 LogonSign=ADMIN,:cs2:cjo:USB0 SetTime=GMT
```

17. Import the Master Backup Key by using the `csadm MBKImportKey` command described in [MBKImportKey](#) (p. 142).
18. The `user.db`, the `cxiskey.db` and, if available, the `auditkey.db` databases have been provided as a prerequisite. Restore them by using the `csadm RestoreDatabase` command.
19. If you are using a LAN device, distribute the `HSMauth.key` file to all computers/persons in your organization shall communicate with the device in a mutually authenticated way.



Your CryptoServer is ready for use again.

## 8 Contact Address for Support Queries

If an error occurs while operating the device, read [\[CSTrSh\] \(p. 243\)](#) to solve it.

If the error still occurs, prepare diagnostic information in a .txt file on your computer as described in [\[CSTrSh\] \(p. 243\)](#).

If you have any further questions on device, feel free to contact us.

You can reach us from Monday to Friday, 09.00 a.m. to 05.00 p.m., Central European Time (CET).

Utimaco IS GmbH  
Germanusstr. 4  
52080 Aachen  
Germany

### RMA Query

If you need to send the device back to Utimaco IS GmbH, please open a new RMA case (Return Merchandise Authorization). We request that you use the following web address. RMA cases cannot be opened by email or phone.

<https://support.hsm.utimaco.com/support/rma/new>

### Other Support Queries

- Mail (preferred contact method)  
[support@utimaco.com](mailto:support@utimaco.com)<sup>1</sup>  
Attach the diagnostic information to your email.
- Web portal  
<https://support.hsm.utimaco.com/support/cases/new/>  
The diagnostic information will be requested in our response if necessary.
- By phone  
AMERICAS +1-844-UTIMACO (+1 844-884-6226)  
EMEA +49 800-627-3081  
APAC +81 800-919-1301  
The diagnostic information will be requested in our response if necessary.

---

<sup>1</sup> <mailto:support@utimaco.com>

## 9 References

<i>Title/Company</i>	<i>Doc.-No.</i>	<i>Location</i>
ANSI X9.63-2001: Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography / ANSI (American National Standards Institute)		
ANSSI (Agence nationale de la sécurité des systèmes d'information): "Avis relatif aux paramètres de courbes elliptiques définis par l'Etat français", Journal officiel de la République française (JORF), n° 0241 du 16 octobre 2011 page 17533 text n° 30 (Announcement about elliptic curve parameters set by the French government). NOR: PRMD1123151V.		<a href="https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000024668816">https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000024668816</a>
RFC 5639, ECC Brainpool Standard Curves and Curve Generation, March 2010.		<a href="https://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf">https://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf</a>
CryptoServer - Firmware Module CMDS - Interface Specification - CMDS Version $\geq 3.0.0.0$ /Utimaco IS GmbH.	2009-0002	Product Bundle
CryptoServer PCIe CSe-Series Operating Manual/ Utimaco IS GmbH.	M013-0002-en	Product Bundle ...Documentation\ Operating Manuals
CryptoServer - CryptoServer CSP and CNG Key Storage Provider/Utimaco IS GmbH.	2008-0002	Product Bundle ...Documentation\ Administration Guides
CryptoServer LAN V5 – Administration Manual/ Utimaco IS GmbH.	2018-0010	Product Bundle ...Documentation\ Administration Guides
CryptoServer LAN V5 Operating Manual/ Utimaco IS GmbH.	2018-0004	Product Bundle ...Documentation\ Operating Manuals
CryptoServer – Administration Manual /Utimaco IS GmbH.	M010-0001-en	Product Bundle ...Documentation\ Administration Guides
CryptoServer – cxitool Manual / Utimaco IS GmbH	2018-0003	Product Bundle ...Documentation\ Administration Guides
CryptoServer – PKCS#11 P11CAT Manual/ Utimaco IS GmbH.	M013-0001-en	Product Bundle ...Documentation\ Administration Guides
CryptoServer –PKCS#11 p11tool2 Reference Manual/ Utimaco IS GmbH.	2012-0014	Product Bundle ...Documentation\ Administration Guides
CryptoServer PCIe Se-Series Gen2 Operating Manual/ Utimaco IS GmbH.	M015-0001-en	Product Bundle ...Documentation\ Operating Manuals

<b><i>Title/Company</i></b>	<b><i>Doc.-No.</i></b>	<b><i>Location</i></b>
CryptoServer Troubleshooting/Utimaco IS GmbH.	M011-0008-en	Product Bundle ...Documentation\ Administration Guides
FIPS PUB 186-4, Digital Signature Standard/National Institute of Standards and Technology (NIST), July 2013.		
PKCS#1: RSA Cryptography Standard v2.1, June 14, 2002/RSA Laboratories.		<a href="https://www.ietf.org/rfc/rfc3447.txt">https://www.ietf.org/rfc/rfc3447.txt</a>
PKCS#3: Diffie-Hellman Key Agreement Standard v1.4, November 1, 1993/RSA Laboratories.		<a href="https://www.teletrust.de/fileadmin/files/oid/oid_pkcs-3v1-4.pdf">https://www.teletrust.de/fileadmin/files/oid/oid_pkcs-3v1-4.pdf</a>
SEC2: Recommended Elliptic Curve Domain Parameters – Certicom Research – January 27, 2010, Version 2.0.		