

u.trust Anchor

Containerized Hardware Security Module (cHSM)

Administration 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	https://support.hsm.utimaco.com/ support@utimaco.com
Document Version	1.0.29
Product Version	6.0.0
Date	2024-10-25
Document No.	2020-0040
Status	PUBLISHED

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

1	Introduction	7
1.1	Target Audience	7
1.2	Document Conventions	7
1.3	Abbreviations	8
2	Overview	10
2.1	Software	11
2.1.1	Operating System	11
2.1.2	Firmware Modules	11
2.1.3	Firmware Module Management	12
2.1.4	FIPS Validated Firmware Packages.....	13
2.1.5	Boot Process	14
2.2	Administration Tools	14
2.2.1	CryptoServer Command-Line Administration Tool (csadm)	15
2.2.2	Administration Tool p11tool2.....	17
3	Concepts.....	19
3.1	Security Mechanisms	19
3.1.1	Alarms	19
3.1.2	Clearing Mechanisms	19
3.1.3	Mutual Authentication	19
3.1.4	Secure Messaging	21
3.2	Smartcards, PIN Pads and Keyfiles	22
3.2.1	Smartcards.....	22
3.2.2	PIN Pads.....	22
3.2.3	Keyfiles.....	23
3.3	User Management, User Groups, and Authentication.....	23
3.3.1	Users	24
3.3.2	User Groups (Standard).....	25
3.3.3	Permissions and Authentication Status	26
3.3.4	User Groups and Access to Commands	29
3.3.5	Maximum Failed Authentication Attempts.....	31
3.3.6	Authentication Mechanisms.....	31
3.3.7	Authentication Modes	35
3.3.8	The Default Administrator ADMIN	35

3.3.9	Default Cryptographic User	37
3.3.10	PKCS#11 Users	37
3.4	cHSM Modes and States	39
3.4.1	cHSM Operating Modes	39
3.4.2	cHSM Operating States	39
3.4.2.1	Retrieving and Understanding cHSM State Indicators	40
3.4.3	Cluster Mode	41
3.5	System Keys	42
3.5.1	Container Master Key	42
3.5.2	Container Admin Authentication Key (CAAK)	43
3.5.3	Master Backup Key (MBK)	43
3.5.4	Audit Log Signature Key	45
3.6	Cryptographic Interfaces	46
3.6.1	Cryptographic eXtended Interface (CXI)	46
3.6.2	PKCS#11	46
3.6.3	Microsoft CryptoAPI and Cryptography API: Next Generation (CNG)	47
3.6.4	Java Cryptography Extension (JCE)	47
3.6.5	OpenSSL	47
3.6.6	Extensible Key Management (EKM)	48
3.7	Interface Hardening by Disabling Selected Functions	48
3.8	Backup of Keys and Users	48
3.9	Database Management	49
3.10	Built-in Elliptic Curves	49
3.11	Supported Algorithms	51
3.11.1	CSP/CNG API	51
3.11.2	CXI API	52
3.11.3	JCE API	54
3.11.4	PKCS#11 API	58
4	Setup	59
4.1	System Requirements	59
4.2	Preparations	59
4.3	Installing the cHSM Host Software	60
4.3.1	Installing csadm	60
4.3.2	Setting up Java Cryptography Extension (JCE)	62

4.4	Device Preparation	63
4.5	Setting up the PIN Pad for Linux	64
4.6	Setting up the PIN Pad for Windows	64
4.6.1	Prerequisites:	64
4.6.2	Installing on Windows 10 and later	65
4.7	Verifying a new cHSM	71
4.8	Claiming a new cHSM	73
4.9	Generating an MBK.....	74
5	Configuration	75
5.1	Setting the CRYPTOSERVER Variable	75
5.2	Using a PIN Pad.....	77
5.2.1	Requirements	77
5.2.2	Configuring the PIN pad Daemon (PPD)	78
5.2.3	Starting the PIN Pad Daemon.....	83
5.2.4	Connecting the PIN Pad to Remote Tools/APIs and a cHSM without a Blocking Firewall	85
5.2.5	Connecting a Local PIN Pad, Remote Tools/APIs, and a cHSM with a Blocking Firewall .	86
5.2.6	Connecting the PIN Pad to Local Tools/API	88
5.2.7	2020-0040a Connecting a PIN pad in a mixed setup.....	89
5.2.8	Authenticating a Command via PIN Pad towards a Remote Computer	89
5.3	Master Backup Key Rollover.....	92
5.3.1	MBK Rollover Preparations	94
5.3.2	MBK Rollover of an Internal Keystore.....	95
5.3.3	MBK Rollover of an External Keystore.....	100
5.4	Configurable Role-Based Access.....	104
6	Monitoring and Maintenance	107
6.1	Indicating a FIPS cHSM.....	107
6.2	Leaving the FIPS Error State	107
6.3	Logs.....	107
6.3.1	Boot Log	107
6.3.2	Audit Log	108
6.3.2.1	Audit Log Entries	109
6.3.2.2	Signed Audit Logs	130
7	Troubleshooting	131
7.1	Alarm Treatment	131
8	Contact Address for Support Queries	132

9 References133

1 Introduction

This manual provides information on the cryptographic usage of Utimaco's u.trust Anchor Containerized Hardware Security Modules (cHSM) running on a u.trust Anchor device.

All u.trust Anchor cHSMs share the same basic functionality. Depending on the template used for its creation, a cHSM can have additional functionalities, like – for example – complying with the approved mode of operation according to [\[FIPS140-3\]](#). Special modes and behaviors for different cHSM functionalities are pointed out in this manual where they apply.

1.1 Target Audience

This document is primarily intended for all persons assuming the cHSM Administrator role for a u.trust Anchor cHSM.

The main task of the **cHSM Administrator** is to set up, administer and maintain the cHSM using the command line tool *csadm*. In other words, the cHSM Administrator provides global configuration services. The main task of the cHSM Security Officer is 'local' (slot or Key Group-specific) configuration and user management.

User Group	Global Configuration Services	Local Configuration Services	Key Management Services	Cryptographic Services	Set 'TRUSTED' attribute of Wrapping Key
cHSM Administrator	✓	✗	✗	✗	✗
Cryptographic User	✗	✗	✓	✓	✗
Key Manager	✗	✗	✓	✗	✗
User	✗	✗	✗	✓	✗
cHSM Security Officer	✗	✓	✗	✗	✓

Table 1: Permissions across User Groups

1.2 Document Conventions

We use the following document conventions:

Convention	Use	Example
Bold	Items of the Graphical User Interface (GUI), e.g., menu options	Press OK
<code>Monospaced</code>	Code that is given for explanation or as an example, file paths	<code>chsm-create</code>

Convention	Use	Example
<i>Italic</i>	References and important terms	See <i>Sample Chapter</i> in the <i>CryptoServer - Sample Manual</i>

Table 2: 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.

1.3 Abbreviations

We use the following abbreviations in this manual:

Abbreviation	Description
AES	Advanced Encryption Standard
BSI	Bundesamt für Sicherheit in der Informationstechnik (Federal Office for Information Security)
CA	Certificate Authority
CAK	Container Authentication Key
cHSM	Containerized Hardware Security Module
CNG	Cryptography API: Next Generation

Abbreviation	Description
CSP	Cryptographic Service Provider
CSR	Certificate Signing Request
CXI	Cryptographic eXtended Interface
csadm	CryptoServer command-line administration tool
DAK	Device Authentication Key
DES	Data Encryption Standard
DKMS	Dynamic Kernel Module Support
DMK	Device Master Key
DRBG	Deterministic random bit generator
DRNG	Deterministic random number generator
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve DSA
EKM	Extensible Key Management
FIPS	Federal Information Processing Standard
GAK	Global Admin Key
GAAK	Global Admin Authentication Key
GIAK	Global Initial Admin Key
gladm	Global Admin Management Tool
JCE	Java Cryptography Extension
JRE	Java Runtime Environment
MAC	Message authentication code
MBK	Master Backup Key
NTP	Network Time Protocol
P11CAT	PKCS#11 CryptoServer Administration Tool
PCIe	PCI Express Interface
PEM	Privacy-Enhanced Mail
PMU	Platform Management Unit
PRNG	Pseudorandom number generator
RSA	Rivest, Shamir, Adleman (cryptosystem)
SDMK	Sticky Device Master Key
TRNG	True random number generator

Table 3: Abbreviations

2 Overview

The u.trust Anchor HSM is a robust, true multi-tenant, converged Hardware Security Module (HSM) platform for payment and general-purpose use cases that enables cloud service providers and enterprises to offer HSM-as-a-Service (HSMaaS). From key generation, storage, management, encryption, signature creation, and key exchange, u.trust Anchor HSMs perform these functions within a tamper-resistant, hardened environment, guaranteeing the integrity and confidentiality of sensitive data.

Based on a modern, container-based solution, the HSM is designed for and inspired by cloud technology, offering up to 31 containers and multiple PKCS #11 partitions per cHSM for application separation and key partitioning. Each instance is opaque, meaning that access to administrative and cryptographic functions is limited to the user, ensuring the required level of confidentiality for their sensitive data and keys. It is easy to manage and gives end users complete control over their keys, with best-in-class performance and auditability.

u.trust Anchor is available in different models that differ in the number of cHSMs that can be created within the u.trust Anchor device.


Low-to-mid performance			Mid-to-high performance 		
cHSMs	RSA 2k performance	Model	cHSMs	RSA 2k performance	Model
1	100	Se100	4	15.000	Se15k
4	2000	Se2k	12	40.000	Se40k
8	5000	Se5k	16	40.000	CSAR 16
			31	40.000	CSAR 31

Figure 1 : u.trust General Purpose (GP) Hardware Security Module (HSM) Se-Series



To upgrade your u.trust Anchor model, your u.trust Anchor device needs to be sent back to Utimaco to perform the upgrade. In-field upgrade licenses will be enabled in a future release.

The u.trust Anchor cHSM is a virtual, protected security module running on a u.trust Anchor device. It is developed specifically to ensure the efficient and secure performance of the following cryptographic operations:

- Generating keys

- Saving keys securely
- Generating random numbers (hardware (true) and software (pseudo) random number generator)
- Generating and verifying signatures
- Calculating hash values

The u.trust Anchor cHSM protects all cryptographic operations and used keys against any form of attack utilizing technical software solutions. The u.trust Anchor cHSM guarantees the trustworthiness and integrity of data within your IT systems.

2.1 Software

Different software components work together within the u.trust Anchor cHSM. This section gives a brief introduction to these software components and describes their functionality.

2.1.1 Operating System

The u.trust Anchor cHSM's operating system is called SMOS Façade (Security Module Operating System). It does not have direct access to the u.trust Anchor cHSM hardware and therefore directs all internal commands that require hardware access to COSMOS, the operating system running on the u.trust Anchor cHSM. It provides mechanisms for task handling, inter-process communication, memory management and file handling. Furthermore, SMOS Façade offers appropriate access command interfaces to the other firmware modules.

2.1.2 Firmware Modules

Firmware modules are an encapsulated software part running inside the u.trust Anchor cHSM. A module can have an external interface which can be used by an application from outside the u.trust Anchor cHSM, and an internal C interface which can be called by other firmware modules.

The firmware modules can be divided into several classes:

- operating system (SMOS Façade),
- standard firmware modules provided by Utimaco
- other application firmware modules

The operating system and standard firmware modules are needed in order to get a running u.trust Anchor cHSM with basic functionality. Other application firmware modules are cryptographic interfaces like PKCS#11 or CXI and are provided by Utimaco.

The functional design of each standard firmware module and its interface is specified in more detail in the respective module-specific documentation, which is available with the SDK (Software Developer Kit) version of u.trust Anchor cHSMs.

With the exception of the command scheduler module CMDS, the administration module ADM, the cryptographic interface module CXI and the Master Backup Key management module MBK, these modules have no external interface and can only be accessed by other firmware modules via an internal public C-Interface.

A u.trust Anchor cHSM generally stores a firmware module in the form of a MSC (module storage container). The MSC format is for u.trust Anchor cHSM-internal use only. It stores the module code together with certain module information and the check value for the module code's integrity (SHA-512 hash value over the module code which will be verified at every start-up).

Failing of Firmware Modules

If the module CMDS or any of its dependencies fail to start, the u.trust Anchor cHSM will enter dead state and not respond to any commands.

If the module CRYPT fails to start, a cHSM will not respond to any commands, while a FIPS cHSM will enter the ERROR state.

If other modules fail to start, the u.trust Anchor cHSM responds to commands but won't register the external interface of the module, i.e. if the module CXI fails to start, the u.trust Anchor cHSM will perform CMDS commands, but CXI commands will not be available and an error message will be returned instead.

2.1.3 Firmware Module Management

It is only possible to load firmware modules provided and signed by Utimaco. It is not possible to load firmware modules from any other source.

With the aim to link the firmware with administrative information (for example, module name and version number) and to add check values for the authenticity and integrity of the firmware, every firmware module is enveloped into a so-called container. These containers allow also the load and storage of encrypted firmware.

Utimaco can supply the u.trust Anchor user with firmware modules signed by Utimaco. They will be provided in `.mtc` format.

- MTC

An MTC (Module Transport Container) is used to secure the transport of the RFM from the developer to the customer and in this way guarantee the authenticity of the RFM during delivery. The MTC adds an MTC header, which contains additional module transport information, and an MTC signature for secure and authentic transport. The MTC header is mandatory but the MTC signature is optional. An MTC is stored as an `*.mtc` file.

- MSC

After the MTC has been transported to and loaded into the u.trust Anchor cHSM, the u.trust Anchor cHSM unwraps the RFM from the MTC and stores it as an MSC (Module Storage Container).

The MSC format is for u.trust Anchor cHSM-internal use only. It stores the RFM together with certain module information and the check value for the RFM's integrity (SHA-512 hash value calculated over the RFM). All sections of an MSC except the type section are mandatory.

Perform the `csadm ListFiles` command to show all available `*.msc` files on a u.trust Anchor cHSM.

2.1.4 FIPS Validated Firmware Packages

The following firmware modules are mandatory for the full functionality of FIPS cHSMs. They are loaded into the cHSM upon its creation by the u.trust Anchor administrator.

Module	File
ADM	adm.msc
AES	crypt.msc
ASN1	crypt.msc
DSA	crypt.msc
ECA	crypt.msc
ECDSA	crypt.msc
HASH	crypt.msc
LNA	crypt.msc
VRSA	crypt.msc
VDES	crypt.msc
CMDS	cmds.msc
CXI	cxi.msc

Module	File
DB	db.msc
FIPS140	fips140.msc
HCE	hce.msc
POST	post.msc
MBK	mbk.msc
SMOS	smos.msc
UTIL	util.msc
CRYPT	crypt.msc

Table 4: Mandatory firmware modules for FIPS cHSMs

2.1.5 Boot Process

After the bootloader has passed the control to the operating system *SMOS Façade*, it will perform the following steps:

- Searching for firmware modules depending on its own starting mode. If SMOS Façade itself has been started, it will start all firmware modules;
- Verifying the integrity of all firmware modules;
- Starting all firmware modules.

If the start of SMOS Façade is successful, the cHSM enters operational mode.

During start-up of SMOS Façade and other firmware modules the cHSM writes log messages to the boot log file.

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

2.2 Administration Tools

The u.trust Anchor cHSM can be administered with two command-line tools, the CryptoServer administration tool *csadm* and the PKCS#11 Administration Tool *p11toolv2*.

The functionality of both administration tools is explained briefly in the following sections. For detailed command references, please refer to the respective manuals, [u.trust Anchor - csadm Manual](#) and [CryptoServer – PKCS#11 p11tool2 Reference Manual](#).



All procedures described in this manual use `csadm`.

2.2.1 CryptoServer Command-Line Administration Tool (`csadm`)

The u.trust Anchor cHSM is administered with the CryptoServer command line tool `csadm`.



This manual describes the general context of `csadm` and the functionality of commands included in specific procedures to administer the cHSM. For a complete overview of all `csadm` installation, syntax, commands and their parameters, refer to the [u.trust Anchor - csadm Manual](#).

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.

The commands to the cHSM will be sent from the host to the cHSM via a TCP connection. Generally, the TCP server ('daemon') running on the device host (which is the computer directly connected to the u.trust Anchor device) forwards incoming commands to the integrated cHSM, but a few commands are responded to by the u.trust Anchor itself (e. g. setting of the TCP timeout).

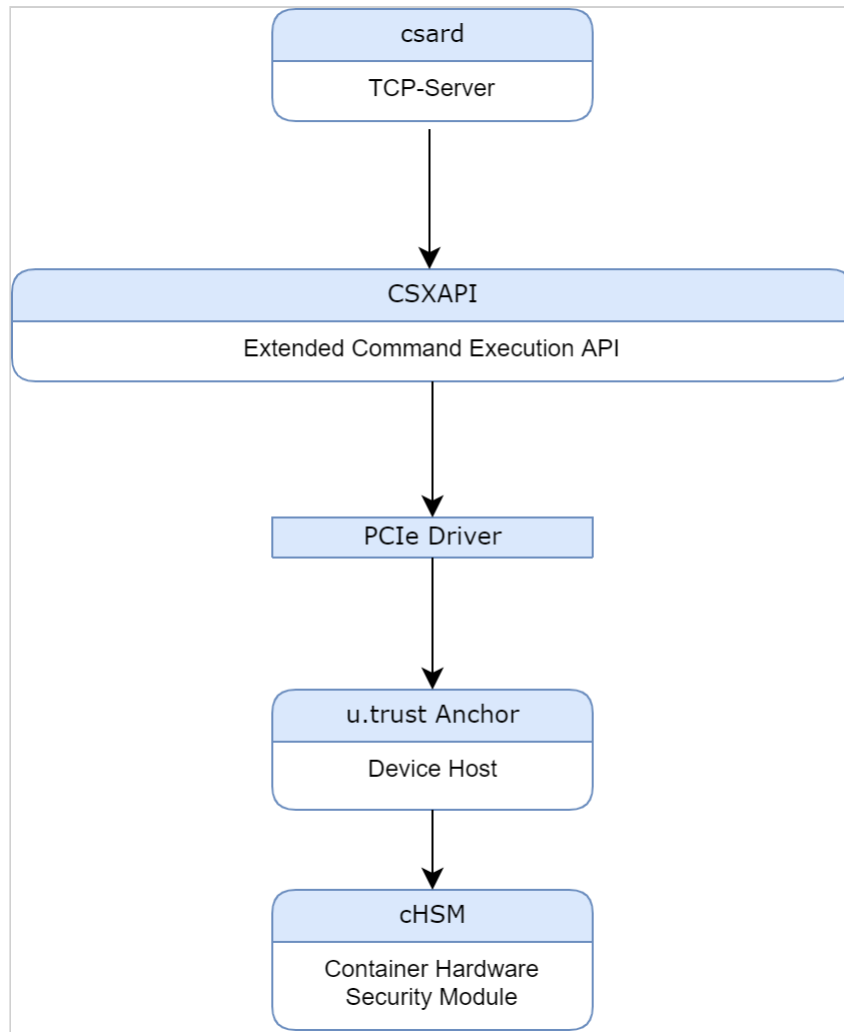


Figure 2 : u.trust Anchor csadm

An application on the host computer can use the extended command execution library CSXAPI to execute any command on a cHSM.

An application on the host computer can use the FIPS-specific library for cryptographic services (CXI library, which also contains the extended command execution library CSXAPI) to execute any of the cryptographic commands which are offered by a FIPS cHSM. CSXAPI (CryptoServer Extended Application Programming Interface) is a software running on the host which has the job to generate a C-interface out of the external cHSM byte stream

interface. (Users who are allowed to assume the role *Cryptographic User* have the right to perform these cryptographic services.)

As a standard application, the Administration Tool *csadm* is available to provide any kind of basic administration such as file management, setting of the cHSM's clock, user management, etc.

Commands can be divided into the following classes:

Command Destination	Counterpart on the cHSM	Command Group
cHSM	Command Scheduler Module (CMDS)	Authentication, User Management
	Administration Module (ADM)	Administration of the cHSM
	MBK Management Module (MBK)	Management of Master Backup Keys
	CXI Module (CXI)	Cryptographic services (not realized in <i>csadm</i>)
TCP server of the device host	Control module of the TCP Server (daemon)	Administration (configuration) of the TCP Server ('daemon')

Table 5: Command classes

2.2.2 Administration Tool *p11tool2*

The *PKCS#11 Administration Tool p11tool2* (*p11tool2*) is a command-line tool designed for being called from the command line or in a batch file. It offers various functions to execute PKCS#11 typical key management and configuration commands on the cHSM. The *p11tool2* is mainly created to support the cHSM's Security Officers and Cryptographic Users or Key Managers when performing PKCS#11 typical tasks:

- An operator who is allowed to assume the cHSM Security Officer role can use *p11tool2* to set up and display group (slot) specific configuration values and to generate or delete the PKCS#11 standard slot User.
- An operator who is allowed to assume the cHSM Administrator role can use *p11tool2* to set up and display global (not group or slot specific) configuration values and to generate or delete the PKCS#11 standard slot Security Officers.
- In the default configuration of the cHSM (configuration attribute `CKA_CFG_AUTH_KEYM_MASK` set to 00000002) every user with permission mask 00000002' (or higher) is allowed to assume the Cryptographic User role (User and Key Manager role) and can use the PKCS#11 administration tool *p11tool2* to execute key management services like key generation or key backup and restore, and to display the current configuration setting.

- If the configuration attribute `CKA_CFG_AUTH_KEYM_MASK` is set to 00000020, key management services can only be executed by dedicated Key Managers (Users with a permission mask of '00000020' or higher). In this case, with p11toolv2 the PKCS#11 standard slot User can only list available keys and storage objects and display the current configuration settings.

The *u.trust Anchor - PKCS#11 p11tool2 - Reference Manual* gives a detailed description of installation and usage of p11toolv2.

If one of the users in the user roles uses the RSA Signature authentication mechanism for authenticating security-relevant PKCS#11 commands, and the private part of his RSA authentication key is stored on a smartcard, the PIN pad must be connected to the administration computer on which the p11tool2 has been installed.

Remarks

- A *slot* according to PKCS#11 corresponds to Key Group `SLOT_<nnnn>`.
- The Security Officer (SO) for slot `<nnnn>` according to PKCS#11 corresponds to user `SO_<nnnn>` with permissions 00000200 (for example, Security Officer `SO_0001` for slot 1 resp. Key Group `SLOT_0001`).
- The slot user according to PKCS#11 for slot `<nnnn>` corresponds to user `USER_<nnnn>` with permission mask '00000002' (for example, user `USER_0001` for slot 1 resp. Key Group `SLOT_0001`).
- Users with different user names, Key Groups or permission masks (for example, Key Managers with permissions 00000020) must be configured by a cHSM Administrator using the appropriate csadm commands.

3 Concepts

3.1 Security Mechanisms

To control and restrict access to security-relevant commands, the u.trust Anchor cHSM implements the concept of users. To perform security-relevant external commands on the u.trust Anchor cHSM, the sender must authenticate the command. Authentication can be done by users who are registered on the device and are equipped with the relevant permissions to authenticate and execute security-relevant commands.

Additionally, the u.trust Anchor cHSM can authenticate itself towards host applications at the beginning of a secure messaging session, if requested by the corresponding application.

3.1.1 Alarms

No alarm can occur directly on the u.trust Anchor cHSM itself. Therefore, it never enters an alarm state.

Alarms occur directly on the u.trust Anchor device. If an alarm occurs for the u.trust Anchor device, all u.trust Anchor cHSMs are stopped and deleted, along with the Device Master Key, which makes all secrets stored on u.trust Anchor cHSMs non-usable since they were encrypted with the Device Master Key. In case your u.trust Anchor cHSM is not reacting to commands, please contact your u.trust Anchor Global Administrator.

3.1.2 Clearing Mechanisms

A u.trust Anchor cHSM can only be cleared by a u.trust Anchor Global Administrator.

3.1.3 Mutual Authentication

Mutual authentication is a security feature that requires that both communicating parties, i.e., the host application and the cHSM device, prove their respective identities to each other before performing any application functions. Security-relevant commands are sent to the cHSM by host applications within a Secure Messaging (see [Secure Messaging](#)) session, that must be always authenticated by one or more users with appropriate permissions.

All u.trust Anchor devices are equipped with a Device Authentication Key Certificate, signed with the Utimaco Root Certificate to verify the authenticity of the device. Upon creation of a cHSM, the device generates a unique Container Authentication Key (CAK). The Global Administrator retrieves the Certificate Signing Request (CSR), which contains the public part of the Container Authentication Key along with additional information, which the Global Administrator gives to the cHSM Administrator. The cHSM Administrator then can sign the

CSR with their Customer Root Certificate Key (or an equivalent certification chain) and can load it into the cHSM as an additional authentication mechanism, see section *LoadCertificate* in the [u.trust Anchor - csadm Manual](#).

When a secure messaging session is established, the answer of the cHSM is signed with the Container Authentication Key (CAK). Additionally, all certificates are returned, which includes:

- the CAK signed with the DAK
- the DAK signed with the Utimaco Root Certificate Key (or an equivalent certificate chain)
- the DAK signed with the Operator Root Certificate Key (or an equivalent certificate chain)
- if available, the CAK signed with the Customer Root Certificate Key of the cHSM Administrator

To use their own certificates for authentication, the cHSM Administrator signs the CAK CSR, loads it into the cHSM and provides every cHSM user with the Customer Root Certificate (consisting of the public key of the root certificate key, signed with the respective private key, (self-signed certificate)). By using the respective parameter when executing `csadm LogonSign` or `csadm LogonPass` (see the [u.trust Anchor - csadm Manual](#)), the cHSM user can verify the certificate chain by using the following parameters in any combination:

- `VendorRootCert=<file>`
- `OperatorRootCert=<file>`
- `CustomerRootCert=<file>`

The connection to the cHSM is then only established when the certificates can be verified. Upon successful verification, the command will be executed without further notifications. In case of failed verification, an error code will be returned.

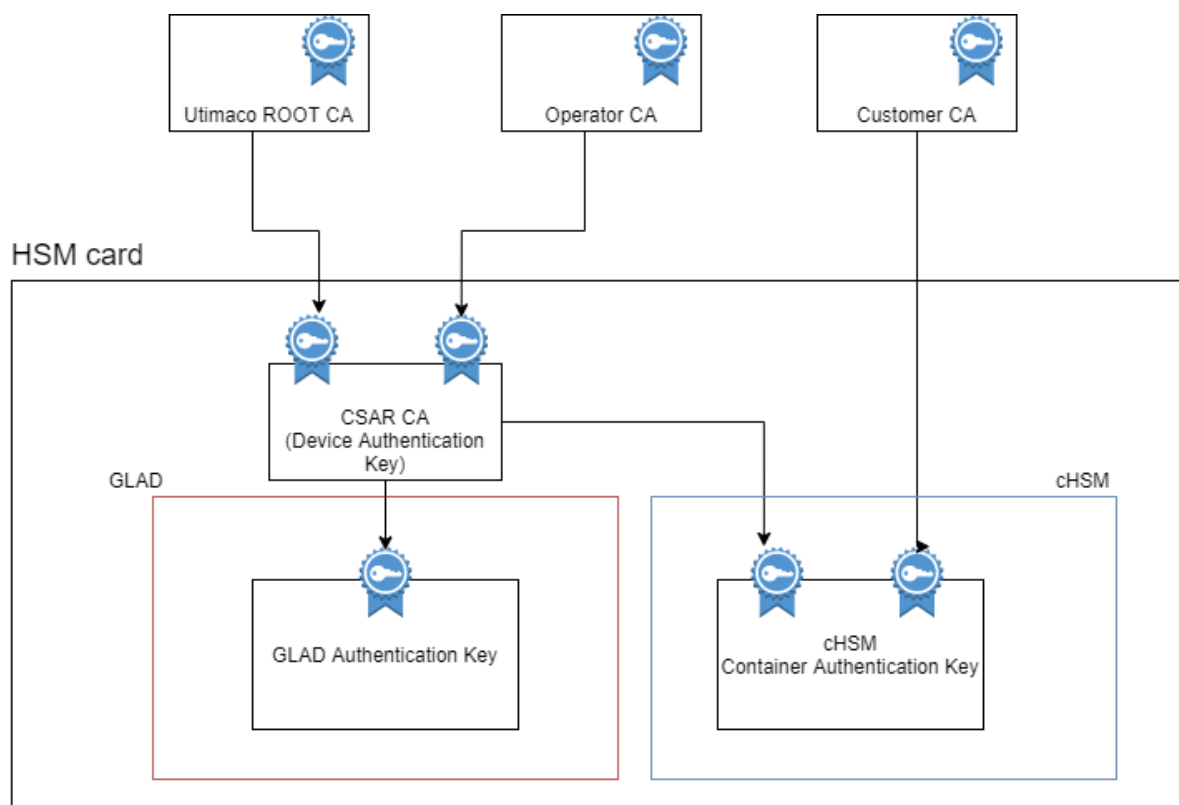


Figure 3 : u.trust Anchor Chain of Trust

3.1.4 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).



A session key automatically becomes invalid if it is not used for 15 minutes. The cHSM 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.

For a detailed description of the Secure Messaging process, see *Secure Messaging* in the [u.trust Anchor - csadm Manual](#).

3.2 Smartcards, PIN Pads and Keyfiles

Smartcards, PIN pads and keyfiles are used to administer the u.trust Anchor cHSM.

3.2.1 Smartcards

The smartcards that can be used are supplied exclusively by Utimaco IS GmbH. You cannot use any other smartcards for administration. The smartcards are preconfigured by Utimaco IS GmbH before they are shipped.



The smartcards are PIN-protected. If an incorrect PIN is entered three consecutive times, the smartcard is blocked and can no longer be used.



Global Administrators should use a dedicated smartcard for operator secrets (and not use the same smartcard for MBKs, for example).

3.2.2 PIN Pads

PIN pads are smartcard readers with an integrated display and a keypad. The PIN pads that can be used with the device are supplied exclusively by Utimaco IS GmbH. You cannot use any other PIN pad.

Utimaco supplies the PIN pad Utimaco cyberJack one.

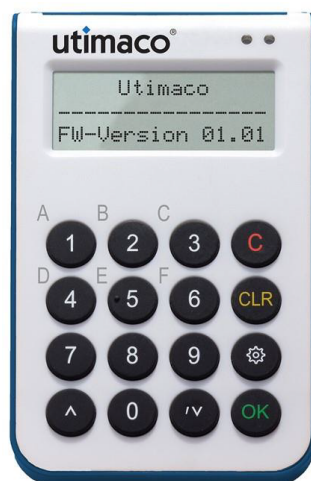


Figure 4 : Utimaco cyberJack one PIN Pad

The PIN pad can be used with smartcards delivered by Utimaco IS GmbH.

3.2.3 Keyfiles

After the installation of the u.trust Anchor host software from the product bundle, the authentication keyfile `ADMIN.key` for the Default Administrator ADMIN is stored on the host computer used for the cHSM administration.

The Global Initial Admin Key (GIAK) is delivered as part of the u.trust Anchor product bundle that can be downloaded from the Utimaco download portal. The GIAK `giak.pem` can be found within the bundle at `/Software/Linux/Administration/key` (for Windows: `\Software\Windows\Administration\key`).



Keyfiles can be used either with or without password protection. We strongly recommend you always use a password to protect your keyfiles.

3.3 User Management, User Groups, and Authentication

To control and restrict the access to security-relevant commands, the u.trust Anchor cHSM implements the concept of users. To perform security-relevant external commands on the u.trust Anchor cHSM, the sender has authenticated the command. Authentication can be

done by users who are registered at the u.trust Anchor cHSM and are equipped with the relevant permissions to authenticate and execute security-relevant commands.

Additionally, the u.trust Anchor cHSM can authenticate itself towards host applications at the beginning of a secure messaging session, if requested by the corresponding application.

If an authentication attempt fails, the respective user is blocked from another authentication attempt for 4 seconds.

3.3.1 Users

The u.trust Anchor cHSM provides the appropriate commands to create or delete a user or to change their authentication token/data with the `csadm AddUser`, `csadm DeleteUser` and `csadm ChangeUser` commands.

For every authentication to be performed, the user name and the authentication token must be specified. Further necessary data depends on the authentication mechanism of the user.

After a successful authentication, the u.trust Anchor cHSM's authentication state will be augmented by the permissions of the user.

Commands can only be successfully authenticated by authorized users. For the management of these users, they are stored and administrated in a user database. In this database for each user the following data will be stored:

Data	Description
Name	Unique identifier for the user
Permission	User's permissions for the different user groups
Authentication mechanism	The authentication mechanisms that must be used by the user
Authentication token	Keyfile or password depending on the authentication mechanism
Attributes	Additional data used to specify particular properties or restrictions, such as that a user may only access specific keys

Table 6: User Data

3.3.2 User Groups (Standard)

The user management functionality has eight user groups, numbered from group 0 to group 7. Some user groups are reserved by Utimaco for applications and their corresponding role-based user profiles.

Group	Application	Role-based user profile and permissions
0	All cryptographic interfaces	Cryptographic User and PKCS#11 User; 00000002
1	PKCS#11	PKCS#11 Key Manager; 00000020
2	PKCS#11	PKCS#11 Security Officer (SO); 00000200
3		
4		
5	NTP Administration	NTP Manager; 00200000
6	System Administration	System Manager; 02000000
7	User Management	User Manager; 20000000

Table 7: User groups and role-based user profiles reserved for specific applications

It is possible to assign new permissions to a predefined user group. As a result, all users in a predefined user group are automatically given the new permission. Therefore, a new permission in group 5 (NTP Administration) can also administrate NTP. As the predefined user groups 6 and 7 have already been assigned wide-ranging administration permissions, we recommend not assigning other permissions to these groups.



The permissions within applications have already been defined at the programming stage. Once an application has been loaded into the device, the assignment to a user group or the permission level cannot be changed. When you assign new permissions within the individual user groups, you must also take into account the fact that several user groups have already been predefined by Utimaco.

Creating a cHSM Administrator

To create a user that is entitled to authenticate all commands for cHSM system administration (e.g. commands for MBK management or management of the cHSM audit log), this user must get permission level 2 in user group 6 or higher (i.e. user permission '02000000' or higher).

Creating a cHSM User Administrator

To create a user that is entitled to authenticate all commands for cHSM user administration (e.g. commands for creating or deleting cHSM users), this user must get the permission level 2 in user group 7 or higher (i.e. user permission '20000000' or higher).

3.3.3 Permissions and Authentication Status

The u.trust Anchor cHSM works with permission levels ranging from 0 (no permission/authentication) to 15 (highest level of permission/authentication).

Permission	Meaning
0	The user has no permissions in that particular user group.
1	Two-Person rule: although the user can authenticate commands to the cHSM, a second user is required for full authentication
2	The user is entitled to authenticate commands on their own.
3-15	Not required by the cHSM firmware functions by default. Can be required by functions with custom permissions defined in a signed configuration file <code>cmds.scf</code> .

Table 8: User permissions

Every user registered on the cHSM is assigned a permission level for each of the existing eight user groups. The permission level of the user defines which permissions can be reached effectively by one or more authentications, determining which commands the user is allowed to execute on their own, which commands need a second user for full authentication, and which commands cannot be performed at all. In communication with the cHSM, the permission level defines which permission must be present before the cHSM can perform a particular command. After an authentication has been performed successfully by the user, the authentication status is increased by the user's permission, starting from authentication status 00000000.

User Group	Role-based User Profile	Auth. Status	Description
0	Cryptographic User and PKCS#11 User	0000002	Permission level 2 in user group 0 is required to perform all external functions implemented by the CXI firmware module which offer cryptographic services with keys and Storage Objects. If the permission mask of the Key Manager role is also set to 00000002 (default), Users are also allowed to perform all key management services.
1			

User Group	Role-based User Profile	Auth. Status	Description
2	PKCS#11 Security Officer	0000020	Permission level 2 in user group 2 is required to perform commands for key group-specific user management, working on Local Configuration Objects, and setting the <code>CXI_PROP_TRUSTED</code> attribute of every assigned wrapping key. The commands for key group specific configuration and PKCS#11 user management provided by the p11toolv2 can only be performed by a Security Officer with permissions 00000200 who is logged in to the cHSM.
3			
4			
5	NTP Manager	0020000	Permission level 2 in user group 5 is required to perform NTP administration functions The NTP module enables the time synchronization on the cHSM by using an NTP server over a network. By default, it is deactivated for use and can only be activated by an NTP Manager.
6	System Manager	0200000	Permission level 2 in user group 6 is required to perform commands for the administration of firmware modules.
7	User Manager	2000000	Permission level 2 in user group 7 is required to perform commands for user management.

Table 9: User groups and role-based user profiles reserved for specific applications

Example for a User's Authentication Status

In this example, the cHSM has four users who have the following permissions.

User Group	7	6	5	4	3	2	1	0
ADMIN	2	2	0	0	0	0	0	0
Admin1	0	1	0	0	0	0	0	0
Admin2	0	1	0	0	0	0	0	0
User3	0	0	0	0	0	0	1	0

No user has been authenticated yet, so the authentication status is 00000000. To perform commands for user management, authentication status 2 has to be reached in user group 6. From the current authentication status 00000000, the following authentication scenarios are possible:

- **ADMIN authenticates**

The authentication status is updated to 22000000. The commands used for user management can now be performed because authentication status 2 has been reached in groups 6 and 7.

- **Admin1 authenticates**

The authentication status is updated to 01000000. The commands used for user management cannot be performed because authentication status 2 has not been reached in group 6.

- **Admin1 and Admin2 authenticate**

The authentication status is updated to 02000000. The commands used for user management can now be performed because authentication status 2 has been reached in group 6. The commands involved in user management cannot be performed because neither of the two administrators has the appropriate permission in group 7.

- **Admin1 and User3 authenticate**

The authentication status is updated to 01000010. Despite the fact that two authenticated users are now present, the commands used for user management cannot be performed because authentication status 2 has not been reached in group 6.



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 for Key Group Restrictions in Authentication Status

User Group	7	6	5	4	3	2	1	0
User1	0	0	0	0	0	0	0	1
User2	0	0	0	0	0	0	0	1

User1 has authentication status 0x00000001 and is assigned to the key group A.
User2 has authentication status 0x00000001 and is assigned to the key group B.

Three cryptographic keys are created.

Key	Key Group	Assigned User
KeyA	A	User1
KeyB	B	User2
KeyAB	AB	

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

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

Commands cannot be performed that require

- 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.

3.3.4 User Groups and Access to Commands

Depending on the application, the eight different user groups/permissions can be used to control the access to sensitive commands.

For each command in an application firmware module, it can be individually defined (as part of the source code implementation) whether an authentication is necessary to perform this command, and if yes, which authentication status is mandatory. The command implementation decides which user group is allowed to access the command and which authentication level within this user group must be reached at least.

Using these access control mechanisms – and if it is wanted for security reasons – it is, for example, possible to determine that a specific command is only available after an authentication following the two-person rule: If, for instance, the necessary authentication status for this command demands authentication level 2 in some user group 'x', and if for this group 'x' only users with permission 1 are registered, then the specific command must be authenticated by two users of this group 'x'. Thus, the command is only accessible if the two-person rule is obeyed.

The implementation of security rules for the authentication of commands within a specific device application depends on the application firmware as well as on the user management.



With the specific implementation of application firmware modules, the framework for the security rules for command authentication is determined. It is fixed which commands must be authenticated and which not. Within this framework of setting rules for user administration, the customer-individual security rules for command authentication can be realized. This can be done by setting the specific permissions and authentication mechanisms for any user when the user is created.

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 which belong to the same key group can be added up.

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 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.

3.3.5 Maximum Failed Authentication Attempts

Within the u.trust Anchor cHSM, 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 the u.trust Anchor cHSM administrator 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 u.trust Anchor cHSM 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 them. Consequently, this user cannot perform any command which must 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*.

3.3.6 Authentication Mechanisms

The u.trust Anchor cHSM supports the following authentication mechanisms:

	<i>HMAC Password Authentication</i>	<i>RSA Signature Authentication</i>	<i>ECDSA Signature Authentication</i>
cHSM	✓	✓	✓
FIPS cHSM	✓	✓	✓

Table 10: Supported authentication mechanisms

HMAC Password Authentication

- **Description**

The u.trust Anchor cHSM supports the HMAC password mechanism for user authentication. The user's password may have a length between 8 and 1024 bytes.

The password is not transferred directly to the u.trust Anchor cHSM but is used as the input for the HMAC key-derived function (HMAC-PBKDF). A random number is added to each authentication. This prevents the authentication from being intercepted and performed again at a later point in time (replay attack).



Adding or restoring HMAC users with MD5 or Rd160 hash algorithms is not supported.

- **Procedure**

First, the host demands an 8-byte random value for each authentication from the cHSM. Then the host calculates the HMAC value over this random value and the command data block using the password as the input for the HMAC key-derived function (HMAC-PBKDF). This HMAC hash value is transferred to the cHSM together with the command data. The cHSM recalculates and checks the hash by using the password stored in the user database. The default hash algorithm for the HMAC calculation is SHA-256. Other hash algorithms can be used on demand for non-FIPS cHSMs.

The HMAC-PBKDF (HMAC password-based key-derived function) 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 fixed 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.

- **Deployment**

This authentication mechanism has the advantage that the password is not submitted in clear text and cannot be scanned. Because of the random value, the authentication data block can also not be scanned and replayed at a later time. A 'Playback'-attack of the command becomes impossible. Additionally, the command data is protected against unnoticed manipulation.

RSA Signature Authentication

▪ Description

The user's authentication token (public RSA key) is stored in the user database of the cHSM. Therefore, the user needs an RSA key pair either on a smartcard or in an optionally encrypted keyfile.



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.

The corresponding public part of the encryption key is stored as authentication data in the cHSM's user database `user.db`. If RSA authentication with a smartcard is used, the smartcard must be inserted into a PIN pad that is connected to a USB port of the computer on which the csadm tool is running.

▪ Procedure

First, the host demands an 8-byte random value from the cHSM. Then the host (or RSA smartcard) calculates an RSA signature over this random value and the command data block with the user's private RSA key (PKCS#11 signature format). This prevents a signed command from being intercepted and entered again later. The RSA signature will then be transmitted to the cHSM together with the command data block. The cHSM will verify the RSA signature with the help of the RSA key's public part stored in the user database `user.db`. The default hash algorithm for the signature calculation is SHA-256. Other hash algorithms can be used on-demand for non-FIPS cHSMs.

▪ Deployment

As RSA signature authentication does not transfer any confidential authentication data, it is particularly suitable for remote authentication to devices when the network connection is not completely under the control of the user.

ECDSA Signature Authentication

▪ Description

The user's authentication token (public ECDSA key) is stored in the user database of the cHSM. Therefore, the user needs an ECDSA key pair either on a smartcard or in an optionally encrypted keyfile.



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

The corresponding public part of the encryption key is stored as authentication data in the cHSM's user database `user.db`. If ECDSA authentication with a smartcard is used, the smartcard must be inserted into a PIN pad that is connected to a USB port of the computer on which the `csadm` tool is running.

▪ Procedure

For this mechanism, the host first demands an 8-byte random value from the cHSM. Then the host (or ECDSA smartcard) calculates an ECDSA signature over this random value and the command data block with the user's private ECDSA key. This command signature will then be transmitted to the CryptoServer which will verify it with the help of the public part of the ECDSA key which is stored in the user database `user.db`. The default hash algorithm for the signature calculation is SHA-256. Other hash algorithms can be used on demand for non-FIPS cHSMs.

▪ Deployment

As ECDSA signature authentication does not transfer any confidential authentication data, it is particularly suitable for remote authentication to devices when the network connection is not completely under the control of the user.

The following table shows which expressions are applied for using the authentication methods in some `csadm` commands.

Authentication mechanism	<i>csadm ListUser</i> output value	<i>csadm authentication</i> commands	<i>csadm user creation</i> input parameter value
HMAC password authentication	HMAC Password	<code>csadm ... LogonPass=...</code>	<code>hmacpwd</code>
RSA signature authentication with a keyfile	RSA Sign	<code>csadm ... LogonSign=...</code>	<code>rsasign</code>
RSA signature authentication with a smartcard			
ECDSA signature authentication with a keyfile	ECDSA Sign		<code>ecdsa</code>

Authentication mechanism	csadm ListUser output value	csadm authentication commands	csadm user creation input parameter value
ECDSA signature authentication with a smartcard			

Table 11: Naming of authentication mechanisms in csadm

The authentication mechanism that is applied to a user can be retrieved from the **Mechanism** column in the output of the `csadm ListUser` command.

Example

Name	Permission	Mechanism	Attributes
ADMIN	22000000	RSA sign	Z[0]
KeyUsrHmacPwd	00000002	HMAC	I[0]
KeyUsrRsaSignKeyfile	00000002	RSA sign	
KeyUsrRsaSignSmartcard	00000002	RSA sign	
KeyUsrEcdsaSignKeyFile	00000002	ECDSA sign	
KeyUsrEcdsaSignSmartcard	00000002	ECDSA sign	

The following table shows the identifiers of the authentication mechanisms. These identifiers are shown in some audit log entries, for example, when an authentication attempt has succeeded or failed.

Authentication mechanism	Identifier
HMAC password authentication	4
RSA signature authentication	0
ECDSA signature authentication	5

Table 12: Authentication mechanism identifier

3.3.7 Authentication Modes

Every security-relevant command must be authenticated by an authorized user before it is sent to the cHSM with the `csadm LogonSign` or `csadm LogonPass` command, depending on the user's authentication mechanism.

3.3.8 The Default Administrator ADMIN

On every newly created cHSM, exactly one Default Administrator ADMIN is present to ensure the initial setup of the cHSM. The Default Administrator ADMIN is allowed to perform all standard administration and user management commands.

They can perform the following administrative tasks:

- Create a user
- Delete users
- Create a backup of user data
- Restore user data
- Set time
- Import Master Backup Key
- Delete audit log

ADMIN is thus allowed to assume the Administrator role according to FIPS standards.

The authentication mechanism of ADMIN is *RSA Signature Authentication*. The user permission of ADMIN is '22000000', i.e. authentication level 2 for user groups 6 and 7. The Default Administrator ADMIN has no permissions in user groups 0 to 5. A first private Default Administrator Key for user ADMIN has to be provided to the cHSM administrator, enabling ADMIN to authenticate towards the u.trust Anchor cHSM. Since this Default Administrator Key is customer-individual, it does not necessarily need to be replaced and the default user ADMIN can be kept.



In case the 2-person rule is desired for the administration tasks: replace the user ADMIN with two (or more) individual users who have the necessary user permissions.

The user permissions in the Default Administrator ADMIN's user groups 6 and 7 cannot be changed. The User Authentication Key used by the default administrator ADMIN to log in to the cHSM can be changed with the `csadm ChangeUser` command. If the default administrator ADMIN shall be replaced, the sum of the permissions for user group 7 of all those of the newly created users who use RSA Signature Authentication mechanism has to be at least 2. u.trust Anchor cHSM allows the deletion of user ADMIN only if this permission level is reached. For the new administrator, only signature-based authentication mechanisms are permitted, since users with these authentication mechanisms are not erased upon an alarm or external erase occurring on the u.trust Anchor device.

Also, at any later date, the u.trust Anchor cHSM will only allow the deletion of a user with Administrator rights if the sum of the permissions of all remaining users who use the RSA key as an authentication token is still above the minimum level of '21000000' (which is the necessary condition to retain an administrable u.trust Anchor cHSM). Otherwise, the command for user deletion will be rejected with an error code.

3.3.9 Default Cryptographic User

A Cryptographic User with permission 2 in the user group 0 (authentication status 00000002) has been set up on the cHSM by default upon its initialization. The Cryptographic User has all permissions in user group 0 and can authenticate all commands on their own. The Cryptographic User has been created for the administration of all cryptographic interfaces:

- CXI (Cryptographic eXtended Interface)
- JCE (Java Cryptography Extension)
- CSP/CNG (Microsoft CryptoAPI and Cryptography Next Generation)
- Open SSL (Cryptographic Token Interface), only if the cHSM is connected directly via an ENGINE interface
- EKM (Extensible Key Management)



The permissions assigned to the Cryptographic User in user group 0 have been defined in the CXI firmware module and cannot be changed.

3.3.10 PKCS#11 Users

If PKCS#11 users are applied, see [PKCS#11](#), special restrictions regarding the users have to be considered.

- **cHSM Administrator**

Only the Default Administrator ADMIN or a cHSM Administrator with permission 2 in the user group 7 (20000000) is permitted to initialize the cryptographic tokens or slots by initializing the Security Officer's (SO's) authentication (PIN or password). There are no restrictions regarding the administrator's authentication mechanism.

- **Security Officer**

The Security Officer (minimum permission 2 in user group 2; 00000200) is responsible for configuring the cryptographic token or slots. The Security Officer can change the PIN or password they use for authentication, which was initialized by the default administrator ADMIN or an administrator with minimum permission 2 in user group 7. In addition, only the Security Officer has the right to initialize the PIN or password used to authenticate the user. 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`. For a Security Officer, only HMAC password authentication is supported.

- **User**

A Cryptographic User is called "User" in the PKCS#11 context. The User (minimum permission 2 in user group 0; 00000002; by default permission 2 in user group 0 and 1, 00000022) can generate, load, delete and use certificates or keys for the cryptographic token. This means that, by default, the User is a combination of a key manager and a key user. The User can change the PIN or password they use for authentication, which was initialized by the Security Officer (SO). This User can also be an application that uses the cryptographic token. The name of the User must be `USR_XXXX` with `XXXX` being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from `USR_0000` to `USR_9999`. For the User, only HMAC password authentication is supported.

- **Key Manager** (Configurable)

The role of the User can be split into a key manager role, with permission (00000020) to manage cryptographic keys, and a key user role, with the permission (00000002) to use cryptographic keys. The cHSM PKCS#11 library should be configured accordingly if these two rules should be used. It is advisable but not mandatory to use `KM_XXXX` as the name of the key manager with `XXXX` being a 2-byte decimal representation of the key group/PKCS#11 slot ID. The name may range from `KM_0000` to `KM_9999`. There are no restrictions regarding the key manager's authentication mechanism.

PKCS#11 users can be created using the following tools:

- **csadm**

csadm can be used to create the PKCS#11 users. For details, see chapter *AddUser* in the [u.trust Anchor - csadm Manual](#).

- **P11CAT**

P11CAT can be used to create a Security Officer or the User. The requirements regarding the name and the authentication mechanism are fulfilled automatically. For details, see the [CryptoServer PKCS#11 - P11CAT Manual](#).

- **p11tool2**

p11tool2 can be used to create a Security Officer (`p11tool2 InitToken` command) or the User (`p11tool2 InitPIN` command). The requirements regarding the name and

the authentication mechanism are fulfilled automatically. For details, see the [CryptoServer - PKCS#11 p11tool2 - Reference Manual](#).

3.4 cHSM Modes and States

3.4.1 cHSM Operating Modes

A functioning u.trust Anchor cHSM is always in Operational Mode. FIPS cHSMs can additionally be in a FIPS error state.

The current mode of the u.trust Anchor cHSM can be retrieved with the `csadm GetState` command. If the u.trust Anchor cHSM does not answer to the `csadm GetState` command, it is in Power Down Mode. In all other modes, the `csadm GetState` command can be performed.

Mode	Description
Operational Mode (optionally Administration-Only)	The u.trust Anchor cHSM is in Operational Mode if its operating system SMOS Façade and further firmware modules were started successfully and are active. The u.trust Anchor cHSM can be explicitly configured to boot into restricted Operational Mode (Operational Mode – Administration-Only). In this special kind of Operational Mode, all cryptographic functions are blocked and only administration functions can be executed, see <i>SetAdminMode</i> in the <i>u.trust Anchor - csadm Manual</i> . If the u.trust Anchor cHSM is not set to boot into Administration-Only Mode via the <code>csadm SetStartupMode</code> command, restarting the u.trust Anchor cHSM will enable full Operational Mode again.
Operational Mode - Administration-Only	The u.trust Anchor cHSM can be explicitly configured to boot into restricted Operational Mode. In this special kind of Operational Mode, all cryptographic functions are blocked, and only administration functions can be executed. A cHSM that is in Operational Mode – Administration-Only can be set back to the standard Operational Mode again, and vice versa. Standard Operational Mode means that the cryptographic functions are no longer blocked.
Power Down Mode	The cHSM is in power down mode if no firmware is active (cHSM is shut down). In power down mode, the cHSM is not able to receive any commands.

Table 13: cHSM Operation Modes

3.4.2 cHSM Operating States

In error-free operation, a u.trust Anchor cHSM is always in state INITIALIZED. It can be used by the cHSM Administrator and assigned users.

Additionally for FIPS cHSMs, if the firmware module CRYPT fails to start, a u.trust Anchor FIPS cHSM can enter the state ERROR state.

```
mode      = Operational Mode
state     = INITIALIZED (0x00100004)
error state (<error code>)
```

To leave the ERROR state, restart the u.trust Anchor cHSM with the `csadm Restart` command. If this does not resolve the issue, please contact your Global Administrator. In state ERROR, many commands are blocked.

Information about the u.trust Anchor cHSM's state can be retrieved via the `csadm GetState` command, see *GetState* in the [u.trust Anchor - csadm Manual](#).

See Availability of csadm commands on FIPS cHSMs for a comprehensive list of commands available in state ERROR.

3.4.2.1 Retrieving and Understanding cHSM State Indicators

For all tasks of the cHSM Administrator, it is vital to be informed about the cHSM's current state and mode.

The `csadm GetState` command returns several parameters that inform about the current state of the cHSM.

<code>csadm GetState</code> output	Meaning
No answer	Dead state or power down mode
mode = Operational Mode state = INITIALIZED alarm = OFF	The cHSM is in initialized state. This implies that the cHSM is generally working and not defective.
mode = Operational Mode state = INITIALIZED FIPS mode = ON alarm = OFF	The cHSM is a FIPS cHSM and is in an initialized state. The cHSM is fully functional according to FIPS standards. Some functionalities are not available for FIPS cHSMs, see (1.0.29) 2020-0040 Availability of csadm commands on FIPS cHSMs.
mode = Operational Mode state = INITIALIZED ERROR FIPS mode = ON alarm = OFF	The cHSM is a FIPS cHSM and has reached an ERROR state. An ERROR state can be returned when FIPS self-tests, like e.g. the conditional test for Random Number Generation, fail. In the ERROR state, many commands are blocked, see (1.0.29) 2020-0040 Availability of csadm commands on FIPS cHSMs for a comprehensive list of commands available in ERROR state.

<i>csadm GetState output</i>	<i>Meaning</i>
mode = Operational Mode - Administration Only state = INITIALIZED alarm = OFF	The cHSM is operational, but the available functionalities have been restricted by the Administration Only mode. All cryptographic services are blocked and must be activated separately, see cHSM Modes and States .

Table 14: Possible state indicators in the output of the csadm GetState command

For more information on handling the cHSM states, see [Troubleshooting](#).

3.4.3 Cluster Mode

u.trust Anchor cHSMs can be part of a cluster.

A cluster of cHSMs can be created by the Global Administrator by cloning a cHSM from a previously made snapshot. This snapshot includes all users and internal keys existent on the cHSM the snapshot is created from. All cHSMs that are created from the same snapshot file are immutable and belong to the same cluster.

To find out if several cHSM belong to a cluster, their UID has to be compared, which is the identifier for the cluster that is listed for each cHSM when using the `csadm GetState` command.



In Cluster mode, no user keys can be stored internally. This affects all CXI commands that generate, derive, or import keys.

Cluster mode does not allow any operations that would result in the internal states of the cHSMs going out of sync. The following functions are blocked in cluster mode:

- Adding Users
- Deleting Users
- Changing Users
- Restoring Users
- Setting Maximal Authentication Failures
- Setting the Admin Mode
- Setting the Startup Mode
- Loading Certificates

- Loading Files
- Deleting Files
- Setting the Time
- Setting the Audit Configuration
- Splitting Keys

3.5 System Keys

Different system keys are used in and around a u.trust Anchor cHSM. The following sections describe how they are generated, who is responsible for storing them, and their intended use. The user authentication keys and the keys in the user's key database (`CXIKEY.db`) are not described herein.

3.5.1 Container Master Key

The Container Master Key is a u.trust Anchor cHSM-individual key that will never leave the u.trust Anchor cHSM.

Key attribute	Description
Type	32 bytes AES key
Generation	Generation of the key will be done automatically by u.trust Anchor, as part of the u.trust Anchor cHSM creation process. It is derived from the Device Master Key, which is the only key present on a u.trust Anchor device. Upon the creation of a new u.trust Anchor cHSM, a Container Base Key is generated and encrypted with the Device Master Key. When the u.trust Anchor cHSM is started, the key is derived from it for the runtime of the u.trust Anchor cHSM.
Usage	The Container Master Key is needed to encrypt data inside of the u.trust Anchor cHSM. For this purpose it will be used internally by other firmware modules. The database firmware module (module DB) facilitates the usage of the key. This DB module provides an internal public interface for secure data storage to other application firmware modules. The DB will use the internal Master Key in full length.

Key attribute	Description
Life Cycle	<p>The Container Master Key cannot be imported but will be generated by u.trust Anchor upon creation of a u.trust Anchor cHSM.</p> <p>After generation, the key is stored inside of the u.trust Anchor cHSM. Key export is not possible. The Master Key will never be available outside of the u.trust Anchor cHSM.</p> <p>The life cycle of this key ends when an alarm occurs to u.trust Anchor and the u.trust Anchor cHSM is deleted as a consequence.</p>

Table 15: Container Master Key

3.5.2 Container Admin Authentication Key (CAAK)

As soon as possible after taking a cHSM into operation, the Container Admin Authentication Key (CAAK) must be used to perform the first administrative tasks, including the set up of a customer-individual cHSM.

Key attribute	Description
Type	RSA key of at least 512 bits size
Generation	<p>Prior to the generation of the cHSM, customer-individual credentials in the form of the public part of a key pair given from the cHSM Administrator to the Global Administrator, who then uses this as initialization data to create a new cHSM using these credentials. Since the private part never leaves control of the cHSM Administrator, it is ensured that only they ever have access to the cHSM.</p> <p>The key is customer- and cHSM-individual.</p>
Usage	The CAAK (Container Admin Authentication Key) is the cHSM Admin's public key file provided to the u.trust Anchor Global Administrator after the creation of a cHSM.
Life Cycle	<p>The key is imported into the cHSM by the Global Administrator upon creation.</p> <p>One copy of the public part of the Container Admin Authentication Key is stored in the user database as the default authentication token of the Default Administrator ADMIN.</p> <p>This copy can be overwritten with the command <code>csadm ChangeUser</code>, which has to be authenticated by ADMIN themselves, or deleted with the command <code>csadm DeleteUser</code>, which can only be performed after sufficient further users with administrator rights have been created to avoid lockout.</p>

Table 16: Details about the Container Admin Authentication Key

3.5.3 Master Backup Key (MBK)

The u.trust Anchor cHSM provides secure storage for secret data and private keys. As any perceived attack on the u.trust Anchor device will cause it to permanently delete all the sensitive data and private keys stored on any cHSMs contained on it, we strongly recommend backing up this data or keys so they can be reimported once the alarm has been reset on the u.trust Anchor device. To ensure this backup copy of the sensitive data or private keys is

stored securely even outside the u.trust Anchor cHSM, it is encrypted with the Master Backup Key (MBK).



Secret data and cryptographic keys stored inside the u.trust Anchor cHSM are encrypted with the device-specific u.trust Anchor cHSM Master Key and not with the MBK. The MBK is for secure storage of the backup copy of this data outside the u.trust Anchor cHSM.

Key attribute	Description
Type	The MBK is an AES key with a key length of 16, 24, or 32 bytes.
Generation	For generating an MBK, you can use the <code>csadm MBKGenerateKey</code> command. The MBK is generated by the MBK firmware module. It is split into <code>n</code> key shares that are exported and stored on <code>n</code> smartcards protected with a PIN or in <code>n</code> key files protected with a password. In order to be stored on the u.trust Anchor cHSM, the MBK has to be imported from these smartcards or key files again. Only then is it stored in an internal database.
Usage	The u.trust Anchor cHSM can store up to 256 MBKs in MBK slot number 0 to 255. However, the default MBK slot number on the u.trust Anchor cHSM is 3. This MBK slot is used by our applications. Upon creation of the cHSM, an MBK is auto-generated in MBK slot 3. The function of an MBK is to protect the backup copy of secret data or private keys that are normally stored on the u.trust Anchor cHSM (and which are used for user authentication) from unauthorized access, even when they have to be stored elsewhere.
Life Cycle	<p>The MBK must be imported into the device by using the <code>csadm MBKImportKey</code> command. The MBK is overwritten whenever a new MBK is imported on the same MBK slot.</p> <p>The MBK is permanently deleted when an alarm has occurred on the u.trust Anchor device. Therefore, you must store a backup copy of this key outside the u.trust Anchor cHSM, for example:</p> <ul style="list-style-type: none"> ▪ On a set of at least two smartcards protected with a PIN. ▪ On a secure USB flash drive or a computer as a set of keyfiles that are protected with a strong password. For security reasons, each MBK share must be assigned to a different person and stored on a different place. <p>The authorized users with the Administrator role shall ensure that the smartcards on which the MBK-shares are stored or the MBK keyfiles and the backup copies of the MBK are securely stored and appropriately protected against unauthorized access, loss, and damage. The PIN codes for these smartcards and the passwords for the keyfiles must be kept secret.</p>

Table 17: Details about the Master BackupKey

3.5.4 Audit Log Signature Key

The Audit Log Signature Key is used to create a signed audit log file. This key is stored in the `auditkey.db` database.

Type	<p>Depending on the <code><mode></code> parameter on creation, the audit log signature key is either an RSA key or an ECDSA key.</p> <ul style="list-style-type: none"> ▪ 3072-bit RSA key with the public exponent 0x10001, PKCS#1 RSA PSS signature with SHA-256 and a 32-byte salt length ▪ ECDSA with NIST P-256 curve and SHA-256
Generation	<p>The Audit Log Signature Key can be created only by performing the <code>csadm GenerateAuditLogKey</code> command.</p>
Usage	<p>The key can be used as input for the <code>csadm GetSignedAuditLog</code> and <code>csadm VerifySignedAuditLog</code> commands.</p> <p>If you want to use a different Audit Log Signature Key, use the <code>csadm DeleteFile=</code> command to delete the <code>auditkey.db</code> database/key. The database is also deleted if an alarm has occurred, or if an External Erase or a clear has been performed.</p> <p>The <code>auditkey.db</code> database can only contain one Audit Log Signature Key.</p> <p>If you want to use a different Audit Log Signature Key, use the <code>csadm DeleteFile=</code> command to delete the database/key and perform the <code>csadm GenerateAuditLogKey</code> command to generate a new key in a new <code>auditkey.db</code> database.</p> <p>Consider that if you delete the <code>auditkey.db</code> database, you cannot sign any audit log files with the key stored in this database anymore and that audit log files previously signed with this key cannot be verified unless you have backed up the <code>auditkey.db</code> database.</p> <p>The <code>auditkey.db</code> database can be backed up and restored using the <code>csadm BackupDatabase</code> and <code>csadm RestoreDatabase</code> commands.</p> <p>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 <code>csadm BackupDatabase</code> command and the <code>csadm DatabaseRestore</code> command.</p> <p>The <code>csadm GenerateAuditLogKey</code> command is supported in Operational Mode only. Perform the <code>csadm GetState</code> command to retrieve the mode of the device.</p>

Life cycle	The Audit Log Signature Key is permanently deleted when an alarm has been triggered, a clear has been performed or a <code>csadm DeleteFile=auditkey.db</code> command has been performed. Therefore, you must store a backup copy of this key outside the device performing the <code>csadm BackupDatabase</code> command and restore it.
-------------------	--

Table 18: Details about the audit log signature key

3.6 Cryptographic Interfaces

This section gives a brief introduction to the cHSM's most important cryptographic interfaces. The corresponding libraries for these interfaces are provided within the u.trust Anchor product bundle.

3.6.1 Cryptographic eXtended Interface (CXI)

The Cryptographic eXtended Interface (CXI) is a proprietary interface developed by Utimaco IS GmbH. CXI provides all cHSM cryptographic functions via a user-friendly interface. It has a low protocol overhead and is easy to integrate into customer-specific applications.

All other cryptographic interfaces and their functions within the cHSM can be controlled via the CXI interface, which is why the CXI interface plays a more prominent role than the other cryptographic interfaces. To administer all cryptographic interfaces including CXI, the Cryptographic User has sufficient permissions. On the cHSM itself, CXI functions are provided by the CXI firmware module. Cryptographic Users can be created to function both as a CXI user and as a user for all other cryptographic interfaces.

3.6.2 PKCS#11

PKCS#11 is the standard used to define a programming interface for security tokens such as smartcards or hardware security modules. From the PKCS#11 viewpoint, a security token is a device that stores objects and can perform cryptographic functions. These objects may be keys or certificates. In addition, the objects can each have different attributes, which not only define how they are handled but can also limit the areas in which they can be used. For these purposes, the tool P11CAT is available.

3.6.3 Microsoft CryptoAPI and Cryptography API: Next Generation (CNG)

Software developers can use the Microsoft cryptographic application programming interface (CryptoAPI) to call powerful cryptographic functions from their Windows-based applications. These functions are provided by what is known as the Cryptographic Service Provider (CSP). The CryptoAPI contains all functions necessary for encrypting and decrypting data (with both symmetrical and asymmetrical keys) and for using and managing digital certificates for authentication purposes.

Cryptography API: Next Generation (CNG) is Microsoft's latest cryptographic platform. In CNG the provider concept has been extended further than in CryptoAPI. CNG also supports newer cryptographic algorithms, like elliptic curves. Both CryptoAPI and CNG are supported, to enable existing applications to be upgraded step-by-step. The CSP/CNG cryptographic interface is accessed via the CXI firmware module. The CSP and CNG interfaces are configured via a dedicated configuration file, and a CSP/CNG user with the same permissions as the Cryptographic User (min. 00000002) must be created manually.

The CryptoAPI and CNG functions are made available to the applications that call them by a CryptoServer Cryptographic Service Provider (CSP) which serves both these interfaces. Additionally, a dedicated configuration file, `cs_cng.cfg`, can be used to configure the CSP/CNG cryptographic interface. A key management tool is available: The command-line utility CNG tool for managing CNG cryptographic keys.

3.6.4 Java Cryptography Extension (JCE)

The Java Cryptography Extension (JCE) is a collection of Java packages that implement cryptographic procedures for Java applications. These include, among other things, encrypting and decrypting data and generating keys. These functions are made available by the JCE provider.

On the cHSM, these JCE functions are made available by the CXI firmware module which is a component of the SecurityServer firmware package.

3.6.5 OpenSSL

OpenSSL is a free implementation of the SSL/TLS protocol for Open Source environments which also offers a range of more specialized functions for certificate management and for different cryptographic functions. These various different functions are grouped together in the OpenSSL command-line tool.

The integration of OpenSSL into the cHSM is done indirectly via an OpenSSL PKCS#11 wrapper. If OpenSSL should be integrated indirectly via a PKCS#11 wrapper, the PKCS#11 administration tool P11CAT can be used to initialize a PKCS#11 token or slot.

3.6.6 Extensible Key Management (EKM)

An SQL Server provides functions for data encryption and EKM (Extensible Key Management). The Microsoft Cryptography API service provider is used for encryption and key generation. Encryption keys for encrypting data and keys are created in temporary key containers, and must be exported by the service provider before they can be saved in the database. This approach enables SQL Server to be used for key management with an encryption key hierarchy and key security.

3.7 Interface Hardening by Disabling Selected Functions

The cHSM offers the possibility to disable selected functions. The firmware functions to be disabled are defined in the custom signed configuration file `cmds.scf`. The configuration file is read and evaluated on every startup of the cHSM. If a function is listed in this configuration file, it is blocked. For detailed information about signed configuration files and their use, see *Using the Signed Configuration File `cmds.scf`* in the [u.trust Anchor - csadm Manual](#). A list of all firmware functions that can be disabled is provided there as well.

3.8 Backup of Keys and Users

In many applications – for security reasons – there is a need to create a backup of the cryptographic keys that are stored in a u.trust Anchor cHSM, and/or to create a backup of the registered users and their user data including their authentication data (public key or password).

To support backup functionality, a u.trust Anchor cHSM provides the possibility to use Master Backup Keys:

A Master Backup Key may be used to encrypt secret data which is exported from the u.trust Anchor cHSM (e.g. create a key backup), or to decrypt data which is imported into the u.trust Anchor cHSM (e.g. restore a key backup). This will e.g. be done by Utimaco's application firmware module CXI.

It can also be used to back up and restore a user and his user data.

On all u.trust Anchor cHSMs, an MBK named `AUTO-GEN` has been autogenerated in MBK slot 3 to make the use of CXI possible immediately. As long as this MBK is in MBK slot 3, certain backup commands are blocked. Another MBK can be imported into MBK slot 3, in which case the autogenerated MBK is automatically moved to another slot. This allows the transfer of keys between CryptoServers and u.trust Anchor cHSMs, or between different clusters of u.trust Anchor cHSMs. Consider that the automatic move only takes place if the

autogenerated MBK is in MBK slot 3. If the autogenerated MBK is in another MBK slot and another MBK is imported to this MBK slot, the autogenerated MBK is overwritten. Other MBKs are always overwritten, if another MBK is imported to the MBK slot it is in.



For backup purposes, it is possible to use an imported MBK to ensure an independent restore possibility.



The handling of Master Backup Keys is implemented over the u.trust Anchor cHSM firmware module MBK. The csadm commands for MBK management are described in *Master Backup Key Commands* in the [u.trust Anchor - csadm Manual](#).

3.9 Database Management

For the databases contained within the u.trust Anchor cHSM, the following functionalities are available:

- Storing a copy of the cHSM databases on a computer (backup),
- Importing a copy of the databases from a backup directory to the cHSM (restore),
- Copy the databases from one cHSM and then import them to a different cHSM (clone).



To use database management functions, an MBK is required. Upon creation of a cHSM, an MBK is autogenerated in slot 3. If databases are to be transferred from one cHSM to another, both cHSMs must have the same MBK.

3.10 Built-in Elliptic Curves

The CXI firmware module of your device can use elliptic curves for ECDSA signature creation, EdDSA signature creation and ECDH key agreement. Each curve is specified by elliptic curve domain parameters and is identified by a name.

The following table lists all built-in elliptic curves by their name, denotes the bit size of their domain parameters (i.e. the key length) and references the specification where the respective curve and its domain parameters are defined.

On high-performance HSM models, numerous common elliptic curves are implemented in hardware.

Name(s)	Size	Defined in	FIPS Approval
NIST-K163 / sect163k1	163	[FIPS186-4], [ANSI-X9.62], [SEC2]	Legacy Allowed for Signature Verification only
NIST-B163 / sect163r2	163	[FIPS186-4], [ANSI-X9.62], [SEC2]	Legacy Allowed for Signature Verification only
NIST-P192 / secp192r1	192	[FIPS186-4], [ANSI-X9.62], [SEC2]	Legacy Allowed for Signature Verification only
brainpoolP224r1	224	[BP]	Allowed
brainpoolP224t1	224	[BP]	Allowed
NIST-P224 / secp224r1	224	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-K233 / sect233k1	233	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-B233 / sect223r1	233	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
curve25519	255	[RFC 7748] with Errata ID 4730	Allowed for key agreement, not allowed for signatures
edwards25519	255	[RFC 7748] with Errata ID 4730	Not allowed
brainpoolP256r1	256	[BP]	Allowed
brainpoolP256t1	256	[BP]	Allowed
NIST-P256 / secp256r1	256	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
secp256k1	256	[SEC2]	Allowed
FRP256v1	256	[ANSSI]	Allowed
NIST-K283 / sect283k1	283	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-B283 / sect283r1	283	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
brainpoolP320r1	320	[BP]	Allowed
brainpoolP320t1	320	[BP]	Allowed
brainpoolP384r1	384	[BP]	Allowed

Name(s)	Size	Defined in	FIPS Approval
brainpoolP384t1	384	[BP]	Allowed
NIST-P384 / secp384r1	384	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-K409 / sect409k1	409	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-B409 / sect409r1	409	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
curve448	448	[RFC 7748] with Errata ID 4730	ECDH only in FIPS mode
edwards448	448	[RFC 7748] with Errata ID 4730	Not allowed
brainpoolP512r1	512	[BP]	Allowed
brainpoolP512t1	512	[BP]	Allowed
NIST-P521 / secp521r1	521	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-K571 / sect571k1	571	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved
NIST-B571 / sect571r1	571	[FIPS186-4], [ANSI-X9.62], [SEC2]	Approved

Table 19: List of built-in elliptic curves



You can find all sources given in this table in chapter [References](#).

3.11 Supported Algorithms

The following chapters list the supported algorithms sorted by API.

3.11.1 CSP/CNG API

The CSP/CNG API supports the following algorithms:

Algorithm	Key length/Curves [bit]	Interface	
		CSP	CNG
DES	56, 112 and 168	✓	✓
AES	128, 192 and 256	✓	✓
RSA	512 – 16.384 (delta = 8 bit)	✓	✓
ECDSA	NIST-P256, NIST-P384, NIST-P521	✗	✓
ECDH	NIST-P256, NIST-P384, NIST-P521	✗	✓

Table 20: Key algorithms supported by the CSP/CNG API

Algorithm	Key length /Curves [bit]	Interface	
		CSP	CNG
MD5 (non-FIPS only)	128	✓	✓
RIPEMD-160 (non-FIPS only)	160	✓	✓
SHA-1	160	✓	✓
SHA-224	224	✓	✓
SHA-256	256	✓	✓
SHA-384	384	✓	✓
SHA-512	512	✓	✓

Table 21: Hash algorithms supported by the CSP/CNG API

3.11.2 CXI API

The CXI API supports the following algorithms.

Algorithm	Key length [bit]	Mode	Chaining mode	Padding
DES (non-FIPS only)	56, 112, 168	Encrypt, Decrypt	ECB, CBC	None, PKCS#5, ISO7816, Random
		MAC	CBC, CBC-Retail	None, Zero, PKCS#5, ISO7816, Random
		Wrap, Unwrap	ECB, CBC	None, PKCS#5, ISO7816, Random

Algorithm	Key length [bit]	Mode	Chaining mode	Padding
AES	128, 192, 256	Encrypt, Decrypt	ECB, CBC, GCM, OFB, CTR, CCM, KWP	None, PKCS#5, ISO7816, Random
		MAC	CBC	None, Zero, PKCS#5, ISO7816, Random
			CMAC	None, Zero, PKCS#5, ISO7816, Random
			GMAC	None
RSA	512 .. 16384 (delta = 1 bit)	Wrap, Unwrap	ECB, CBC, OFB, KWP	None, PKCS#5, ISO7816, Random
		Encrypt, Decrypt	-	None, PKCS#1-v1_5, PKCS#1-OAEP
		Sign, Verify	-	PKCS#1-v1_5, X9.31, PKCS#1-PSS
ECDSA ECDH	Brainpool: 160 ... 512 NIST: 163 .. 571 Secp: 112 .. 571 FRP256v1	Wrap, Unwrap	-	None, PKCS#1-v1_5, PKCS#1-OAEP
		Encrypt, Decrypt (ECIES)	-	-
		Sign, Verify (according to ANSI X9.62; format: Raw or ASN.1)	-	-
EdDSA	Edwards25519 Edwards448	Sign, Verify	-	-
		Wrap, Unwrap	-	-
DSA (non-FIPS only), DH (non-FIPS only), DH_PKCS (see info box below table; non-FIPS only)	P: ≥ 512 Q: ≥ 160	Sign, Verify	-	-
RAW (Generic Secret)	80 ... 8192 (delta = 8 bit)	Hash Computation (HMAC) Sign, Verify (HMAC) Derive Key		

Table 22: Key algorithms supported by the CXI API



DH_PKCS mandatorily only contains the prime P and the generator G as domain parameters. Q is optional. DH mandatorily contains all three DSA domain parameters P, Q, and G.

If random padding bytes are needed, the deterministic random number generator (DRNG) is always used.

Algorithm	Hash length [bit]
SHA-1	160
SHA-224	224
SHA-256	256
SHA-384	384
SHA-512	512
SHA3-224	224
SHA3-256	256
SHA3-384	384
SHA3-512	512

Table 23: Hash algorithms supported by the CXI API

3.11.3 JCE API

The JCE API supports the following algorithms:

Algorithm	Padding Modes
SHA1withRSA	PKCS1, PSS
SHA224withRSA	PKCS1, PSS
SHA256withRSA	PKCS1, PSS
SHA384withRSA	PKCS1, PSS
SHA512withRSA	PKCS1, PSS
SHA3-224withRSA	PKCS1, PSS
SHA3-256withRSA	PKCS1, PSS
SHA3-384withRSA	PKCS1, PSS
SHA3-512withRSA	PKCS1, PSS
MD5withRSA	PKCS1, PSS
SHA1withECDSA	-
SHA224withECDSA	-
SHA256withECDSA	-

Algorithm	Padding Modes
SHA384withECDSA	-
SHA512withECDSA	-
SHA3-224withECDSA	-
SHA3-256withECDSA	-
SHA3-384withECDSA	-
SHA3-512withECDSA	-
MD5withECDSA	-
SHA1withDSA	-

Table 24: Signatures algorithms supported by the JCE API

KeyPairGenerator

Algorithm
RSA
EC
DSA

Table 25: KeyFactory algorithms supported by the JCE API

KeyFactory

Algorithm
RSA
EC
DSA

Table 26: KeyFactory algorithms supported by the JCE API

SecretKeyFactory

Algorithm
DES
DESede
AES

Table 27: SecretKeyFactory algorithms supported by the JCE API

KeyStore

Algorithm
CryptoServer

Table 28: KeyStore algorithms supported by the JCE API

SecureRandom

Algorithm
SHA1PRNG
CryptoServer

Table 29: SecureRandom algorithms supported by the JCE API

KeyGenerator

Algorithm
DES
DESede
AES

Table 30: KeyGenerator algorithms supported by the JCE API

Cipher

Algorithm	Block mode	Padding mode
DES	ECB, CBC	NONE, PKCS5, ISO10126
DESede	ECB, CBC	NONE, PKCS5, ISO10126
AES	ECB, CBC, OFB	NONE, PKCS5, ISO10126
AES	GCM, CCM	NONE
RSA	-	NONE, PKCS1, OAEP

Table 31: Cipher algorithms supported by the JCE API

MAC

Algorithm	Block mode	Padding mode
DES	CBC	NONE,
DESwithPKCS5Padding	CBC	PKCS5

<i>Algorithm</i>	<i>Block mode</i>	<i>Padding mode</i>
DESwthISO10126Padding	CBC	ISO010126
AES	CBC	NONE
AESwthPKCS5Padding	CBC	PKCS5
AESwthISO10126Padding	CBC	ISO010126

Table 32: MAC algorithms supported by the JCE API

<i>Algorithm</i>	<i>Mode</i>
DES	HmacMD5
DES	HmacSHA1
DES	HmacSHA224
DES	HmacSHA256
DES	HmacSHA384
DES	HmacSHA512
DES	HmacSHA3-224
DES	HmacSHA3-256
DES	HmacSHA3-384
DES	HmacSHA3-512
DES	HmacRMD160
AES	HmacMD5
AES	HmacSHA1
AES	HmacSHA224
AES	HmacSHA256
AES	HmacSHA384
AES	HmacSHA512
AES	HmacSHA3-224
AES	HmacSHA3-256
AES	HmacSHA3-384
AES	HmacSHA3-512
AES	HmacRMD160

Table 33: MAC and hash algorithms supported by the JCE API

MessageDigest

<i>Mode</i>
MD5

Mode
SHA-1
SHA-224
SHA-256
SHA-384
SHA-512
SHA3-224
SHA3-256
SHA3-384
SHA3-512
RMD-160

Table 34: Message digest modes supported by the JCE API

3.11.4 PKCS#11 API

For a detailed list of supported PKCS#11 Mechanisms and Functions, please refer to *PKCS#11 Defined Mechanisms and Functions* in the [CryptoServer PKCS#11 R3 - Developer Guide](#).

4 Setup

To set up a new u.trust Anchor cHSM, the following steps need to be taken:

1. Install the software required to install and run the cHSMs on a host computer, see [Installing the cHSM Host Software](#).
2. Optionally provide the Global Administrator with the public part of your Default Administration Key, see [Device Preparation](#).
3. Verify the legitimacy of the new u.trust Anchor cHSM, see [Verifying a new cHSM](#).
4. After verifying the u.trust Anchor cHSM, claim it by uploading your certificate, see [Claiming a new cHSM](#).

4.1 System Requirements

- CPU: Intel x86/x64, AMD x86/x86-64
- Hard disk capacity: at least 120 Mbyte
- RAM: more than 12 Mbyte
- Network card: Ethernet 10/100/1000 Mbit/s
- Operating systems: A complete list of supported operating systems is provided in the release notes.
- Current Java versions for Windows/Linux

4.2 Preparations

The u.trust Anchor cHSM Administrator must provide the Global Administrator with the public portion of the Container Admin Authentication Key (CAAK) for the Default Administrator ADMIN.

If the cHSM Administrator needs to generate the key and extract the public part first, they can do so with the following example:

```
csadm GenKey=myADMIN.key,2048,chsm_admin  
generating RSA key: myADMIN.key, 2048 bits, owner: chsm_admin  
csadm PubKey=myADMIN.key SaveKey=myADMIN.pub
```

4.3 Installing the cHSM Host Software

In this chapter, we describe how you install the software you need to run the cHSMs on a host computer with Unix/Linux operating system. This includes:

- The CryptoServer administration tool csadm
- The CryptoServer Crypto APIs (JCE, CXI, CSP/CNG, PKCS#11)
- The complete documentation of the cHSM

Some preparations are necessary before the software for administering the cHSM is installed.

4.3.1 Installing csadm

Installing csadm on a computer with a Windows operating system

This section describes how to install csadm on a Windows administration computer. csadm is used to manage cHSMs.

On an administration computer running a Windows operating system, csadm is installed by default during the installation of the u.trust Anchor software provided in the product bundle.

The following steps describe how to install csadm on a Windows host computer.

The `csadm.exe` file for Windows can be found in the product bundle here:

- For Windows 32-bit operating systems
Not supported
- For Windows 64-bit operating systems
`Software\Windows\Administration\`

1. Copy the `csadm.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`, which sets the CryptoServer address permanently (unless a `Dev=` parameter is explicitly set for a specific command).



csadm has been successfully installed on the administration computer.

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

This section describes how to install csadm on an administration computer. csadm is used to manage CHSMs.

You find the `csadm` file in the product bundle here:

- For 32-bit operating systems
Not supported
- For 64-bit operating systems
`/Software/Linux/Administration/`

1. Create a `~/bin` directory in your user directory, if there is not one yet:

```
mkdir ~/bin
```

2. Copy the csadm relevant for your operating system into the `~/bin` directory. An example for Linux 64-bit:

```
cp <mount point of the product CD>/Software/Linux/Administration/csadm ~/bin
```

3. Ensure that you have write and execute permissions for csadm.

```
chmod -R u+w+x ~/bin
```

4. Add the `~/bin` directory to the path in the user configuration file in the shell that is being used. In this example, a bash is used as the shell, i.e., open the `~/.bashrc` file and add the following line to it:

```
export PATH=$PATH:~/bin
```

5. Save the changes and close the `~/.bashrc` file.



csadm has been successfully installed on the administration computer.

An environment variable CRYPTOSERVER can be set to avoid setting the `Dev=` parameter for every command execution (e.g., with the value `/dev/cs2.n.m` where `n = 0` and `m = {1, 2, 3, 4}` or `m = {1, 2 ..., 12}`). The CRYPTOSERVER variable defines the cHSM address permanently, but is ignored when a `Dev=` parameter is explicitly set for a specific command.

4.3.2 Setting up Java Cryptography Extension (JCE)

To use the JCE interface, the Java(TM) Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files need to be present. They can be downloaded from the Oracle website.



The Java version installed on the computer can be checked by performing the `java -version` command before the download of the Java(TM) Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files is started.

1. Load the appropriate ZIP file `jce_policy-x.zip`.
2. Extract `jce_policy-x.zip`.
From this file, the files `local_policy.jar` and `US_export_policy.jar` are required. The following example shows the path for Java 6 on a computer with a Linux operating system and 64-bit architecture.
`/usr/lib/jvm/java-6-openjdk-amd64/jre/lib/security/
US_export_policy.jar
/usr/lib/jvm/java-6-openjdk-amd64/jre/lib/security/local_policy.jar`
3. Copy the policy files to the appropriate directory on the Linux system.



JCE has been set up successfully.



In the `.../Software/JCE/CryptoServer.cfg` file, the `ConnectionTimeout` parameter specifies the maximum time in milliseconds to wait before the connection establishment is aborted if the device is not responding.

In practice, the timeout can reach approximately $2 \cdot n \cdot T$.

Legend:

- n: Number of HSMs specified by the `Device` parameter
- T: The timeout value specified by the `ConnectionTimeout` parameter

In the same file, the `Timeout` parameter specifies the maximum time in milliseconds to wait for the answer from CryptoServer after sending a command.

In practice, the timeout can reach approximately $2 \cdot n \cdot T$.

Legend:

- n: Number of HSMs specified by the `Device` parameter
- T: The timeout value specified by the `Timeout` parameter

4.4 Device Preparation

A u.trust Anchor cHSM is created by the Global Administrator. The Global Administrator then provides the necessary files to claim and administer the device to the cHSM Administrator.

Before the creation of a cHSM, the cHSM Administrator must provide the Global Administrator with the public part of their Container Admin Authentication Key (CAAK). This key is then used by the Global Administrator as initialization data for the new cHSM.

Procedure

1. Install the CryptoServer Administration Tool `csadm` as described in [Installing csadm](#).
2. Generate a key with the `csadm GenKey` command and extract the public components.

```
csadm GenKey=myADMIN.key,2048,chsm_admin  
  
generating RSA key: myADMIN.key, 2048 bits, owner: chsm_admin  
  
csadm PubKey=myADMIN.key SaveKey=myADMIN.pub
```

3. Send the public part of the key to the Global Administrator.



The Global Administrator will use the Container Admin Authentication Key to create the new u.trust Anchor cHSM.

4.5 Setting up the PIN Pad for Linux

There is no PIN pad driver installation required for host computers with a Linux operating system. The definition of a special `udev` rule is necessary.

```
echo 'SUBSYSTEM=="usb", ATTRS{idVendor}=="0c4b", ATTR{idProduct}=="0600",  
MODE="666"' > /lib/udev/rules.d/z80_cyberjack.rules
```

For PIN pad details, see [PIN Pads](#).

4.6 Setting up the PIN Pad for Windows

To be able to use the delivered PIN pad for the cHSM administration, the appropriate driver must be manually installed on the host computer where the host software has already been installed.

For PIN pad details, see [PIN Pads](#).



When performing the `csadm LogonSign` command, the following error message might be shown:

```
Error B91D0003  
    PIN pad API  
    no device found
```

In this case, install the PIN pad driver.



When administering the cHSM (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.

4.6.1 Prerequisites:

- You have uninstalled any previous PIN pad driver versions on the host computer.

- You have installed the product CD or you have inserted it into the computer's CD-ROM drive.



Under no circumstances should you install the "Base components" USB driver for Windows from the company REINER SCT.

If your Windows operating system already has REINER SCT's "Base Components" driver installed, it is essential that you remove it before you install the Utimaco IS GmbH driver.

The official REINER SCT "Base Components" driver and the driver from Utimaco IS GmbH cannot be used in parallel.



Under no circumstances should you run the "cyberJack Device Manager" from the REINER SCT Base components, and perform a firmware update of the PIN pad driver.

Installing a firmware update on the PIN pad renders this PIN pad unusable with u.trust Anchor administration tools and cryptographic libraries. After a firmware update, it is not possible to revert to the PIN pad firmware as shipped by Utimaco IS GmbH.

4.6.2 Installing on Windows 10 and later

1. Connect the USB plug of the PIN pad with the host computer.

If you are prompted in a separate dialog box to download the PIN pad driver from the Internet or to let Windows install it automatically, close this dialog box without selecting any of these options.

2. Start the device manager.

- a. Open the **Run** dialog by pressing the Windows key and then simultaneously pressing the **R** key.

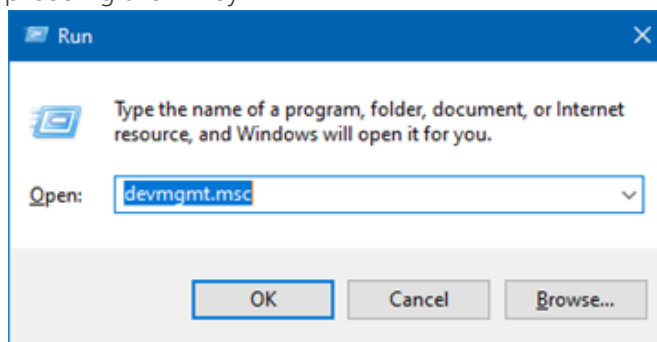


Figure 5 : Run dialog box

- b. Enter `devmgmt.msc` into the text field and press **ENTER**.
The Windows Device Manager opens.

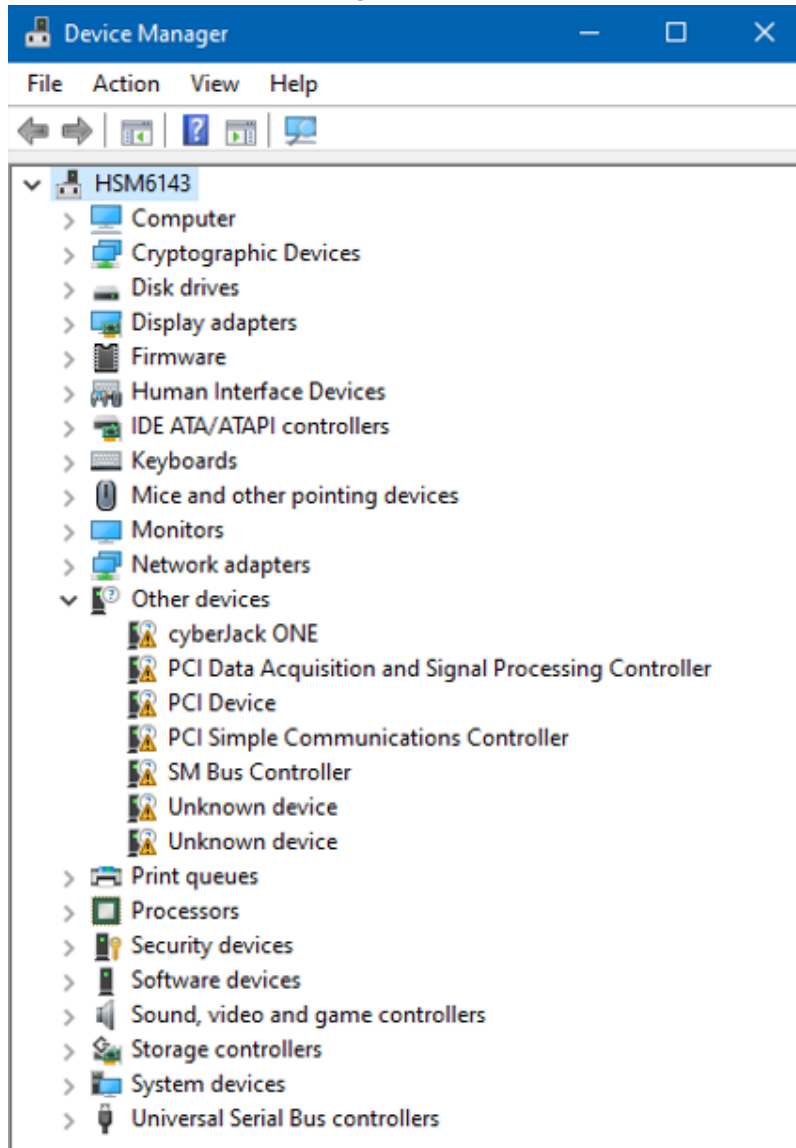


Figure 6 : Windows Device Manager

3. Double click the **Other Devices** entry in the displayed list.
4. Depending on the PIN pad you are using, proceed as follows:
 - a. The Utimaco cyberJack one PIN pad has been supplied to you. Double-click **cyberJack ONE**.

The **cyberJack ONE Properties** dialog box opens.

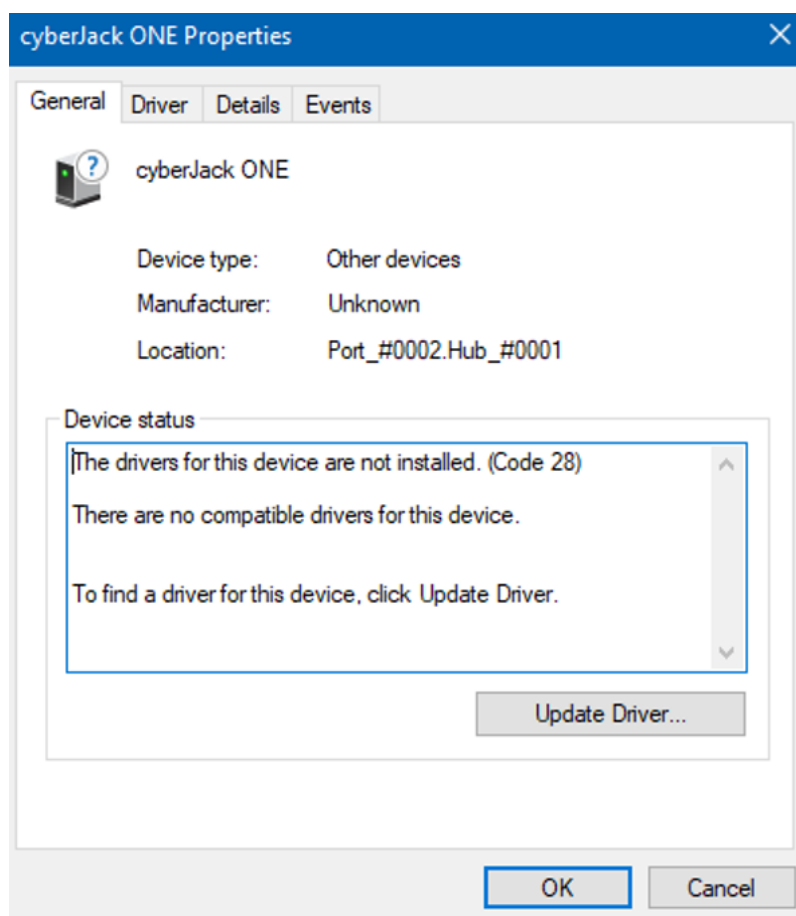


Figure 7 : Dialog box for PIN pad driver installation

5. Click **Update Driver**.

The **Update Drivers - cyberJack ONE** dialog box opens.

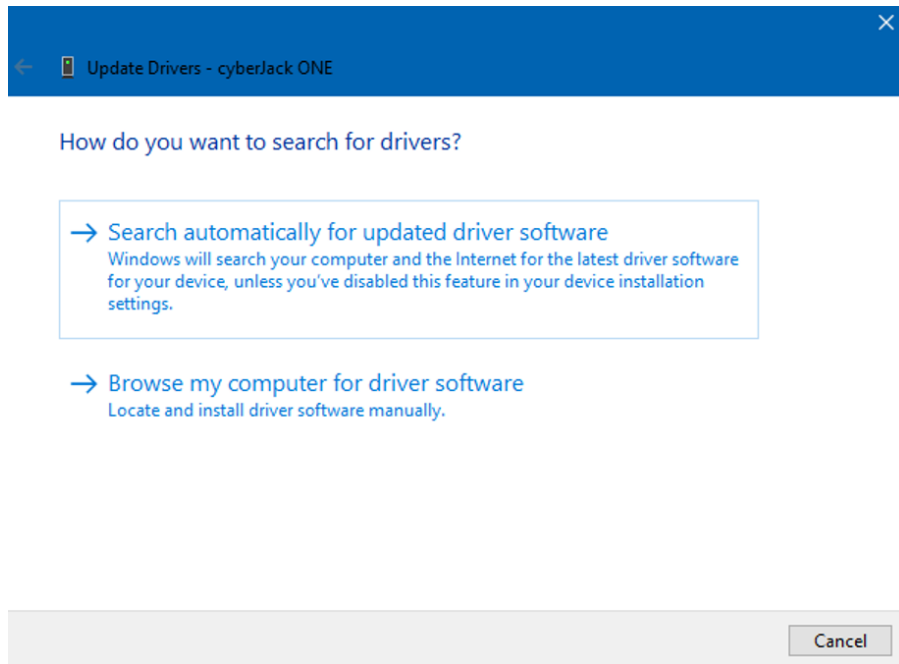


Figure 8 : Dialog box for choosing the PIN pad driver installation

6. Click **Browse my computer for driver software**.

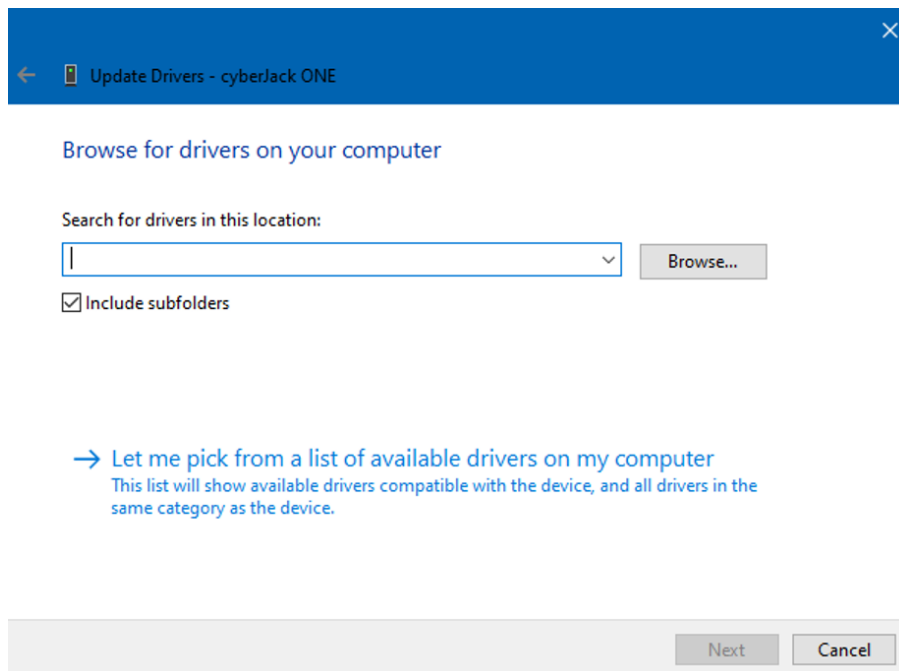


Figure 9 : Dialog box for browsing for drivers

7. Click **Browse** and select the PIN pad driver.
If the installation of the PIN pad driver had been selected in the installer, you find the driver in `C:\Program Files\Utimaco\SecurityServer\cyberJack`.
As an alternative, you find the driver software on the product CD delivered by the Utimaco IS GmbH here: `\Software\Windows\Tools\cyberJack\Driver\`
8. Click **Next**.
9. If a **Windows Security Warning** dialog opens, click **Install**.
The PIN pad driver is now being installed. When completed, the successful driver installation is confirmed in a separate dialog box.
10. Click **Close** in the **cyberJack ONE Properties** dialog box.

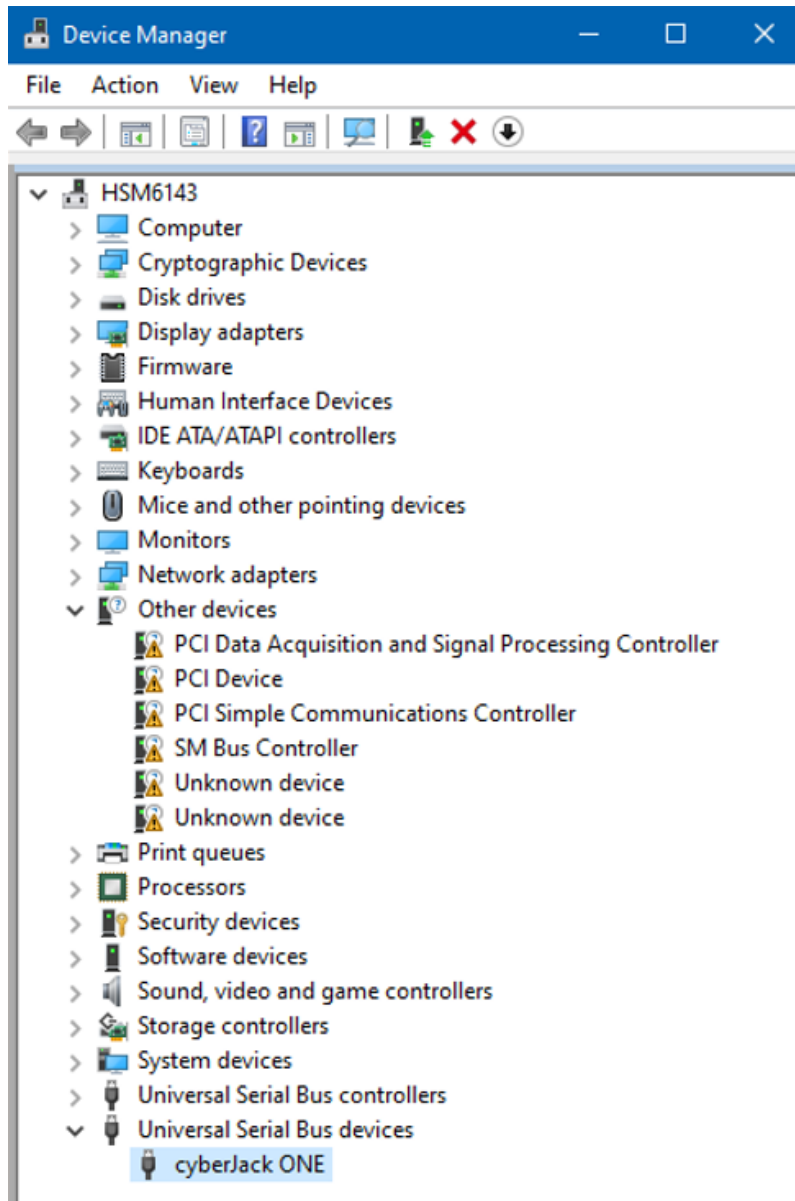


Figure 10 : Windows Device Manager with installed driver



The PIN pad is now displayed with its name **cyberJack ONE** in the Windows **Device Manager** as a subentry of the **Universal Serial Bus devices** entry.

4.7 Verifying a new cHSM

When a new u.trust Anchor cHSM is received, it has to be claimed by the u.trust Anchor cHSM Administrator. In this process, the u.trust Anchor cHSM Administrator will perform checks on the files received by the Global Administrator to verify that the cHSM was created by a genuine Utimaco device with the required keys and certificates.

Prerequisites

To claim a cHSM, the u.trust Anchor cHSM Administrator needs to have received the following files from the Global Administrator:

- the cHSM creation receipt;
- the CSR for the Container Authentication Key (CAK);
- the certificates for the vendor signed certificate of the Device Authentication Key (DAK);
- the DAK signed certificate of the CAK and the GlAD Authentication Key (GAK), and
- the operator signed certificate of the DAK.

```
chsm-receipt_015_20210507121151  
chsm-signature_015_20210507121151  
chsm-auth-key-csr_015_20210507121151  
chsm-auth-key-crt_015_20210507121151  
glad-auth-key-crt_015_20210507121151  
dak-vendor-crt_015_20210507121151  
dak-operator-crt_015_20210507121151
```

Procedure

1. For convenience in further operation, the files provided by the Global Administrator can be renamed as follows:

```
mv chsm-receipt_015_20210507121151 receipt_data  
mv chsm-signature_015_20210507121151 receipt_sig  
mv chsm-auth-key-csr_015_20210507121151 cak.csr  
mv chsm-auth-key-crt_015_20210507121151 cak-dak.pem  
mv glad-auth-key-crt_015_20210507121151 gak-dak.pem  
mv dak-vendor-crt_015_20210507121151 dak-vendor.pem  
mv dak-operator-crt_015_20210507121151 dak-operator.pem
```

2. Check the timestamp of the creation receipt and make sure it's reasonably recent.

```
grep 'Creation time' receipt_data
Creation time: 2021-05-07 10:10:44 GMT
```

3. Check that the receipt is for the correct key.

```
grep 'Modulus' receipt_data
Modulus:
B7F3893AAB150BAFECC1931097893C38751AD728DD56DEB8F1A41097755B5E0664FF32FD902
B04EDCFD5E2EF8330FDF07C15F9C2229E53F71446EEDBC82BEA3D1679B2BBC07B269D0832D0
98B3478189CB1FD9F770ED5231EE9AA05BEBE2D0F13F4813F919EB8B3B14AEEE0EE22EDEB15
2CB5B5798712CDE28273B7E5AB232EB
grep 'MOD' myADMIN.key
MOD=B7F3893AAB150BAFECC1931097893C38751AD728DD56DEB8F1A41097755B5E0664FF32F
D902B04EDCFD5E2EF8330FDF07C15F9C2229E53F71446EEDBC82BEA3D1679B2BBC07B269D08
32D098B3478189CB1FD9F770ED5231EE9AA05BEBE2D0F13F4813F919EB8B3B14AEEE0EE22ED
EB152CB5B5798712CDE28273B7E5AB232EB

# Or use diff:
$ diff <(grep Modulus receipt_data | cut -f 2 -d ' ') <(grep MOD
myADMIN.key | cut -f 2 -d '=') \
  && echo "identical"
identical

$ grep 'Public exponent' receipt_data
Public exponent: 010001
$ grep PEXP myADMIN.key

PEXP=010001
```

4. Check that the cHSM was created with the provided Container Admin Authentication Key (CAAK) by a genuine, untampered Utimaco device.

```
shell
openssl verify -CAfile uti_root.pem -untrusted dak-vendor.pem gak_dak.pem
gak-dak.pem: OK
openssl x509 -in gak-dak.pem -noout -pubkey >gak_pub.pem
openssl dgst -sha256 -verify gak_pub.pem -signature receipt_sig
receipt_data
Verified OK
```

5. Check that we are in the operator's fleet.


```
openssl verify -CAfile operator_root.pem -untrusted dak-operator.pem gak-dak.pem
gak-dak.pem: OK
```

6. Verify the cHSM's CAK.

```
openssl verify -CAfile uti_root.pem -untrusted dak-vendor.pem cak-dak.pem
cak-dak.pem: OK
openssl verify -CAfile operator_root.pem -untrusted dak-operator.pem cak-dak.pem
cak-dak.pem: OK
```

7. Check that the UUID in the receipt and in the CAK certificate's subject DN match.

```
diff <(grep UUID receipt_data | cut -f 2 -d ' ') \
<(openssl x509 -in cak-dak.pem -noout -subject | sed 's/.CN = ([a-f0-9-])/\1/' --) \
&& echo "identical"
identical
```

8. Check that the Certificate Signing Request is for the correct key, namely the CAK.

```
diff <(openssl x509 -in cak-dak.pem -noout -pubkey) \
<(openssl req -in cak.csr -noout -pubkey) && echo "identical"
identical
```



The cHSM has been verified successfully.

4.8 Claiming a new cHSM

Once the Global Administrator has verified that the u.trust Anchor cHSM was created by a genuine Utimaco device with the required keys and certificates, see [Verifying a new cHSM](#), it can be claimed. To claim the device, the cHSM Administrator has to upload their own certificate.



csadm currently only supports loading DER formatted certificates.

1. Upload the certificate of the Container Authentication Key with the `csadm LoadCertificate` command.

```
cat >cak.ext <<EOF
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical,CA:false
extendedKeyUsage = critical,serverAuth
keyUsage = critical,digitalSignature
EOF
openssl x509 -req -in cak.csr -CAkey myCA.key -CA my_own_root.crt -CAform
der \
-out my_cak.crt -outform der -extfile cak.ext
csadm LogonSign=ADMIN,myADMIN.key LoadCertificate=my_cak.crt
```



The u.trust Anchor cHSM has been claimed successfully.

4.9 Generating an MBK

In order to independently create and restore backups, it is necessary to generate a Master Backup Key (MBK). For details on this concept, see [Backup of Keys and Users](#) and [Master Backup Key Rollover](#).

To generate an MBK, use the `csadm MBKGenerate` command as described in *MBKGenerate* in the [u.trust Anchor - csadm Manual](#).

5 Configuration

5.1 Setting the CRYPTOSERVER Variable

For setting up and administering the u.trust Anchor cHSM, the command-line tool `csadm` requires the address of the device in order to connect. This address is specified for every command in the `Dev=` command parameter, see *Syntax of csadm* in the [u.trust Anchor - csadm Manual](#).

Example:

```
csadm Dev=3001@194.168.4.107 GetState
```

Optionally, and for more convenience, the address of the u.trust Anchor cHSM can be set permanently in a 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 `cs_pkcs11_R2.cfg` file
This parameter is used for actions initiated by P11CAT or p11tool2. For details, see *CryptoServer – PKCS#11 P11CAT Manual*
- Device parameter in the `cs_cng.cfg` file
This parameter is used for actions initiated by CSP/CNG. For details, see *CryptoServer - CryptoServer CSP and CNG Key Storage Provider*
- Device parameter in the `csxlan.conf` file
This parameter is used for some communication situations.



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
```

5.2 Using a PIN Pad

csadm can authenticate a user by reading RSA keys that are stored on a smartcard. To do so, the smartcard must be inserted into a PIN pad that is directly connected to the computer csadm is running on. This PIN pad is called a local PIN pad.

u.trust Anchor cHSMs also offer the possibility to connect the PIN pad to a local computer while csadm and the u.trust Anchor cHSM are running on a remote computer.

A mix of these scenarios is also possible. For example, one PIN pad may be connected to the local computer and another one connected to a remote computer.

The following sections give an overview of these scenarios.



The PIN pad daemon is a TCP server that provides access to a PIN pad over the network. It serves remote procedure calls from a remote PIN pad API and dispatches them to a local PIN pad API. It can use secure messaging to encrypt messages to the commands.

The default port number has been set to 6070 but it can be changed in the configuration file.

5.2.1 Requirements

The following requirements must be fulfilled:

Local Computer

- A PIN pad must be connected to the local computer via a USB port.
- The PIN pad driver must be installed, see [Using a PIN Pad](#).
- The authentication key for authenticating a csadm command must be available on a smartcard. Use the `csadm GetCardInfo` command to verify this.
- For the connection to remote tools/API with a blocking firewall, an SSH client, for example, PuTTY, must be installed. If no firewall is present, any similar terminal software client, for example Remote Desktop Connection, is sufficient.

PIN Pad Daemon Files

- Retrieve the PIN pad daemon files from the support team of Utimaco IS GmbH, see [Contact Address for Support Queries](#). These files are:
 - PIN pad daemon executable
 - `ppd.cfg` : Configuration file
 - `ppd.pwd` : Password file

Remote Computer

The necessary u.trust Anchor tools (for example csadm) is installed.

- **Network client**

Netcat or ncat can be used. Check the package manager of your Linux distribution for the package to be installed.
- **SSH Daemon**
 - For the connection to remote tools/API without a blocking firewall, any terminal server software similar to SSH, for example, Remote Desktop Connection, is sufficient.
 - For the connection to remote tools/API with a blocking firewall, an SSH daemon must have been installed.

To apply Scenario 3 with a remote Windows computer, you must install an SSH daemon (for example, OpenSSH) on the remote computer. Installing and operating this SSH daemon is out of the scope of this documentation.
- **Network**

For scenario 2, the port of the terminal software (for example, port 22 for ssh) and port 6070 (default; configurable) must not be blocked by a firewall between the local computer and the remote computer.

For scenario 3, port 22 must not be blocked by a firewall between the local computer and the remote computer.

5.2.2 Configuring the PIN pad Daemon (PPD)

When started, the PIN pad daemon reads its configuration values from the `ppd.cfg` configuration file. To make this happen automatically (i.e. without the configuration file being specified in the start command), the configuration file must be provided in one of the following directories (`<configuration path>`).

`$HOME` (the home directory of the user)

`$HOME/ppd`

`/etc`

`/etc/ppd`

In the configuration file, lines starting with `#` are comments. There are two sections, `[Global]` and `[Server]`. The following items are configured in this file.

▪ PIN Pad Daemon Log File

The PIN pad daemon has its own log file. `LogFile` defines the path to this file.

Sample log entries:

Starting the PIN pad daemon including the configuration parameters

```
PPD is starting...  
...  
PPD is running
```

Terminating the PIN pad daemon.

```
PPD is stopping...
```

Failed authentication attempt in a command that has been received by using the PIN pad daemon.

```
authentication failed
```

Error message of the PIN pad API due to a command that has been received by using the PIN pad daemon.

```
Error B91D501F  
  
PIN pad API  
  
USB  
  
errno = 31
```

A connection has been established to perform a command.

Example: `TCP connection from 192.123.123.123:49333 accepted on port 6070`

Default configuration path value:

<TEMP>/ppd.log, that means, typically /tmp/ppd.lo

- **Logging level**

LogLevel defines which information is logged in the log file defined by the Logfile parameter.

1: Error

2: Warning

3: Info

4: Trace

Default value: 2

- **Log file size**

LogSize is the maximum size of the log file in byte before a new log file is created.

Default value: 8000000

- **Authentication mechanism**

The PIN pad API on the remote computer communicates with the local computer over a TCP connection (default; This can be changed to UDP). If the `AuthMech` parameter in the configuration file is set to 1, this communication is encrypted with secure messaging with an AES-256 key in CBC Mode. In this case, exchanging the session key is authenticated using SHA-512 and with the password defined in the password file that is defined by the Passfile parameter in the configuration file.

If `AuthMech` is set to 0, no secure messaging is done at all.



The authentication mechanism mentioned here must not to be confused with the authentication mechanisms of the device.

- **Password File**

The `Passfile` parameter only applies if the `AuthMech` parameter has been set to 1.

The `Passfile` parameter indicates the path to the password file. This file contains only the Password parameter indicating the password for authenticating the exchange of the session key for secure messaging.

The maximum length of the password is 79. The password must not contain any blanks. The hashtag character (#) in the password is not supported, since it will be interpreted as the beginning of a comment. Other special characters in the password, for example, the following ones, are supported:

Colon: :

Dot: .

Hyphen: -

Comma: ,

The Passfile parameter's default value is `<configuration path>/ppd.pwd` on Linux.

In the following cases, no authentication is done for exchanging the session key of secure messaging. That means an anonymous session is used:

- The password file defined by the Passfile parameter cannot be found.
- The Password parameter in the password file is empty.

If authentication is done for exchanging the session key for secure messaging, the password defined in the password file must be entered when a command is performed by a cHSM tool/API. If no authentication is done, a password must not be entered.

▪ Port Number

The port number the PIN pad daemon listens on for incoming connections.

In scenario 2, this port must be opened in all firewalls between the local computer and the remote computer. In scenario 3, it is not necessary that this port is opened in the firewalls because the firewalls are tunneled by using SSH.

Default value: **6070**

▪ Protocol

The protocol to be used for the communication between the PIN pad daemon on the local computer and the CryptoServer tools/APIs on the remote computer.

Possible values: `TCP` or `UDP`

Default value: TCP

- **IPv6**

IPv6 indicates whether IPv6 is supported. The PIN pad daemon is able to use IPv6 with either TCP or UDP. We recommend using TCP with IPv4.

Possible values: 0: Disabled, 1: Enabled

Default value:

- **Timeout**

Timeout in seconds until an idle connection between the PIN pad daemon and the CryptoServer tool/API is terminated. A timeout value of 0 indicates that an idle connection is never terminated.

Default value: 600

Sample Configuration File

```
[Global]
```

```
# Log file [default: <TMP>/ppd.log]
```

```
# LogFile=/var/log/ppd.log
```

```
# Log level (0:NONE, 1:ERROR, 2:WARNING, 3:INFO, 4:TRACE) [default: 2]
```

```
LogLevel=3
```

```
# Maximum size the log file may grow before rotation [default: 8MB]
```

```
LogSize=1000000
```

```
# Authentication mechanism: 0: NONE 1:AUTH_MECH_SHA512 [default: 1]
```

```
# AuthMech=0
```

```
# Password file for authentication method AUTH_MECH_SHA512
```

```
# [default: <CONFIGPATH>/ppd.pwd]
# Passfile=/etc/ppd/ppd.pwd

[Server]
# Port number [default: 6070]
Port=6070
# Protocol, TCP or UDP [default: TCP]
Protocol=TCP

# Enable IPv6 (0: Disable, 1: Enable)
#IPv6=1 # [default: 0]

# Timeout until an idle connection is closed in seconds [default: 600]
IdleTimeout=60
```

5.2.3 Starting the PIN Pad Daemon

To start the PIN pad daemon on Linux, perform the following steps.

1. Copy the `ppd` file to an arbitrary target directory, for example, `/usr/bin` or `/usr/sbin`.
2. Copy the `ppd.cfg` and `ppd.pwd` files to one of the directories where they are read automatically.
3. To run the PIN pad daemon in the foreground, perform the following command.

If the `ppd.cfg` configuration file is in one of the directories where it is automatically read:

```
ppd -foreground
```

Otherwise:

```
ppd -config=<configuration path> -foreground
```

`<configuration path>` is the path to the `ppd.cfg` configuration file.

Example:

```
ppd -config=/etc/ppd/ppd.cfg -foreground
```

As an alternative, if you want to run the PIN pad daemon in the background – that is, as a daemon – perform the following command.

```
ppd -config=<configuration path>
```

If you need help, perform the `ppd -help` command. It produces the following output:

```
@(#) ppd PinPAD Daemon 1.1.0 [X64] (Sep 14 2017)

valid parameters are:

-config=<path> - explicitly determine config file path
-foreground   - execute PPD without service installation
-version      - show version information
-help         - show this text
```

4. The start of the PIN pad daemon is logged in the log file.

```
19.02.2018 16:09:25 | ppd_init          | @(#) ppd PinPAD Daemon 1.1.0
19.02.2018 16:09:25 | ppd_init          | [X64] (Sep 14 2017)
19.02.2018 16:09:25 | ppd_init          | I: configuration file : /etc/pwd/
ppd.cfg
19.02.2018 16:09:25 | ppd_init          | I: LogFile           : /etc/pwd/
ppd.log
19.02.2018 16:09:25 | ppd_init          | I: LogLevel          : 3
19.02.2018 16:09:25 | ppd_init          | I: LogSize           : 8388608
19.02.2018 16:09:25 | ppd_init          | I: AuthMech          : 1
19.02.2018 16:09:25 | ppd_init          | I: Passfile          : /etc/pwd/
ppd.pwd
19.02.2018 16:09:25 | ppd_init          | I: Port              : 6070
19.02.2018 16:09:25 | ppd_init          | I: Protocol          : 6
19.02.2018 16:09:25 | ppd_init          | I: IPv6              : 0
19.02.2018 16:09:25 | ppd_init          | I: IdleTimeout       : 60
19.02.2018 16:09:25 | ppd_init          | I: PPD is starting...
19.02.2018 16:09:25 | ppd_main_task     | I: PPD is running
```

5. It might be useful to provide start/stop/init scripts.

5.2.4 Connecting the PIN Pad to Remote Tools/APIs and a cHSM without a Blocking Firewall

In this scenario, the PIN pad is directly connected to a local computer. The u.trust Anchor cHSM tools/APIs are installed on a remote computer. Both computers are connected to each other by a local area network. There is no firewall between these computers. You have physical access only to the local computer; that means that any input via a keyboard must be done here. The local computer must provide a method to forward the keyboard input to the remote computer by, for example, using an SSH connection or a remote desktop connection. Following this example, an SSH client on the local computer may open an arbitrary port on this computer and connect it via the LAN to the remote computer where an SSH daemon listens to port 22 and forwards the information to csadm. That is, you open a command-line on the local computer, and in this command-line, you perform a csadm command on the remote computer.

This csadm command communicates via the local area network with a u.trust Anchor cHSM.

If this csadm command must be authenticated by a key that is available on a smartcard in a PIN pad that has been connected to the local computer, csadm opens an arbitrary port on the remote computer and connects it via the local area network to port 6070 (default value) on the local computer where a PIN pad daemon listens to this port 6070. This daemon communicates with the PIN pad via a USB port.

The operating system on the local computer may differ from the one on the remote computer, for example, Windows on the local computer and Linux on the remote computer.

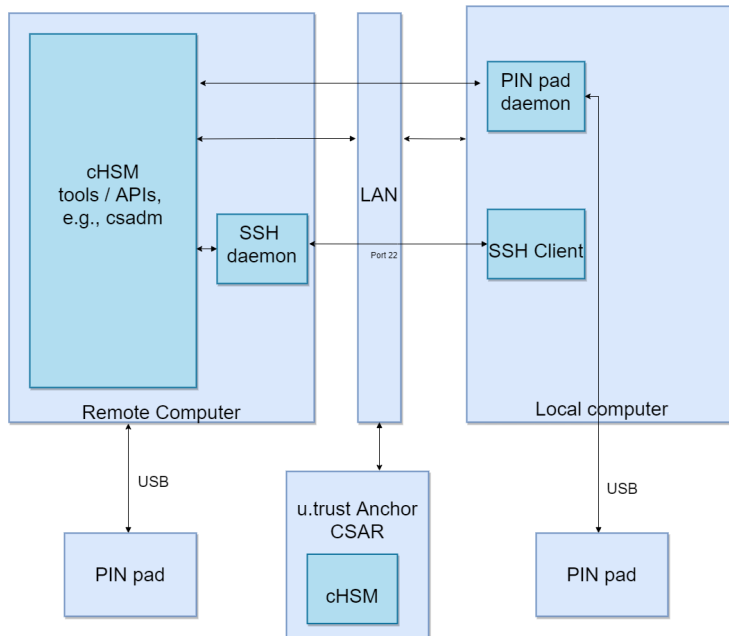


Figure 11 : PIN pad access to a remote cHSM without a blocking firewall

5.2.5 Connecting a Local PIN Pad, Remote Tools/APIs, and a cHSM with a Blocking Firewall

If there are one or more firewalls between the local computer and the remote computer that block port 6070, scenario 3 must be used.

In this scenario, the PIN pad is directly connected to a local computer. The u.trust Anchor cHSM tools and APIs (csadm, for example) are installed on a remote computer. Both computers are connected to each other by a local area network.

You have physical access only to the local computer; that means that any input via a keyboard must be done here. The local computer must provide an SSH client on the local computer. No remote desktop connection or anything similar can be used. The SSH client opens an arbitrary port on this computer and connects it via the LAN to the remote computer where an SSH daemon listens to port 22 and forwards the information to csadm. That is, you open a command-line on the local computer, and in this command-line you perform a csadm command on the remote computer.

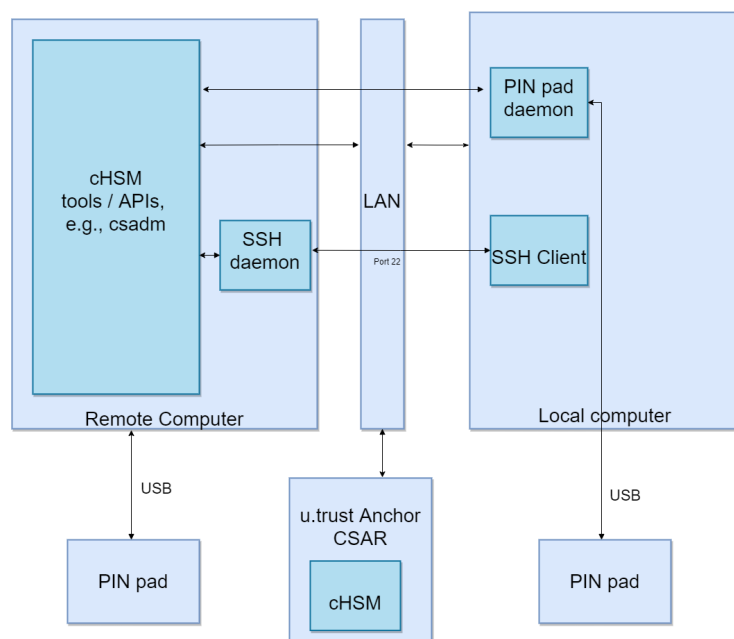


Figure 12 : PIN pad access to a remote cHSM without a blocking firewall

This csadm command communicates either via the local area network with a u.trust Anchor cHSM.

If this csadm command has to be authenticated by a key that is available on a smartcard in a PIN pad that has been connected to the local computer, csadm opens an arbitrary port on the remote computer and connects it to port 6070 (default value) on the same computer. The SSH daemon listens to this port and forwards the connection to the local computer where the SSH client opens an arbitrary port and connects to port 6070 (default value) where the PIN pad daemon listens. This daemon communicates with the PIN pad via a USB port.

The operating system on the local computer may differ from the one on the remote computer; for example, Windows on the local computer and Linux on the remote computer.

The main difference to scenario 2 is that csadm does not communicate directly with the PIN pad daemon via the local area network but it uses the SSH connection between the local computer and the remote computer as a tunnel.

5.2.6 Connecting the PIN Pad to Local Tools/API

In this scenario, the cHSM tools and APIs are installed on a local administration computer. The cHSM is hosted by a remote u.trust Anchor device. They are connected via a local area network. Commands that are triggered by the cHSM tools/APIs can be authenticated by using a PIN pad that is directly connected to the administration computer via a USB port.

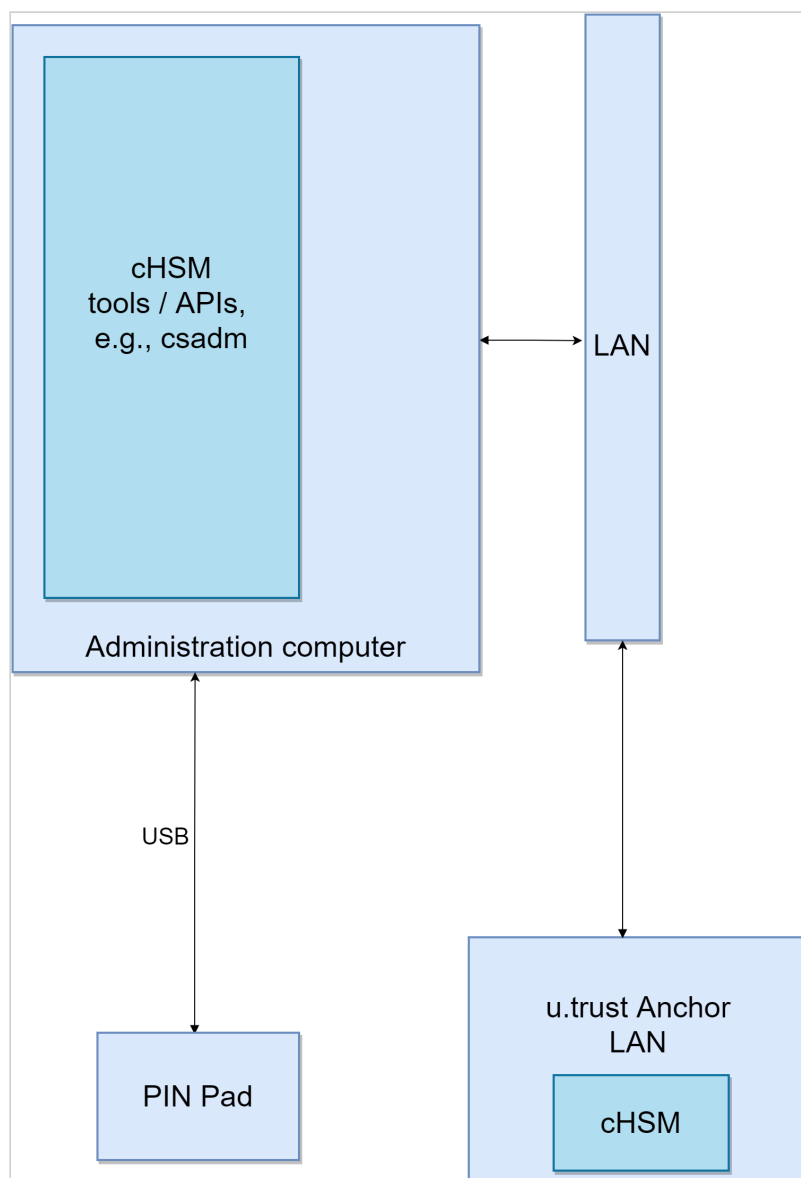


Figure 13 : PIN pad access to a u.trust Anchor LAN

5.2.7 2020-0040a Connecting a PIN pad in a mixed setup

The scenarios described above may also be combined. For example, one PIN pad can be connected to the local computer and another one connected to the remote computer. This might be suitable if a command must be authenticated by more than one user (two-person rule). The following figures show two examples.

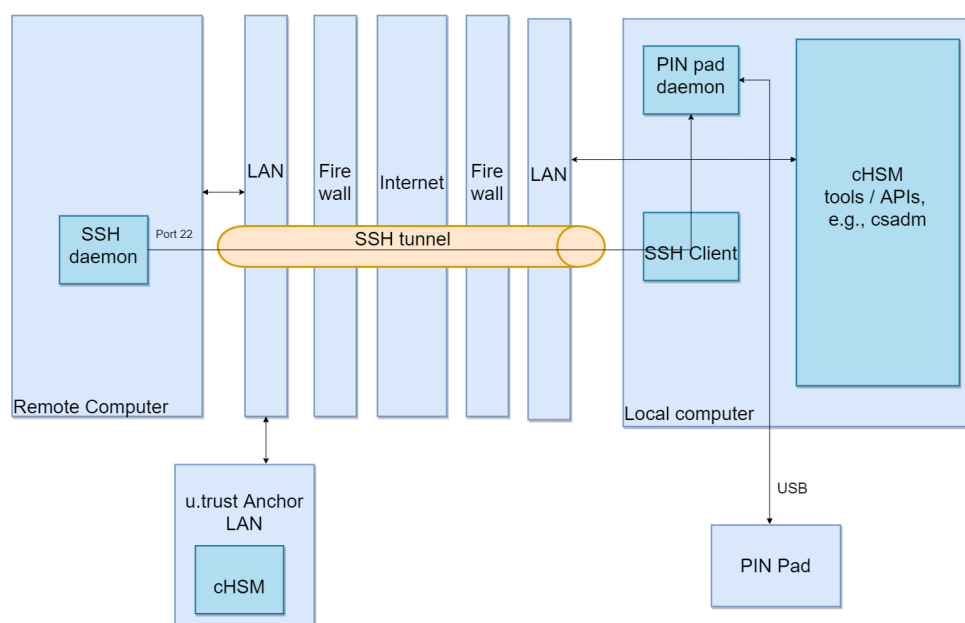


Figure 14 : An example for a mixed scenario

5.2.8 Authenticating a Command via PIN Pad towards a Remote Computer

This section describes the syntax for how to authenticate a command that is performed on a remote computer by a PIN pad that is connected to a local computer. The complete syntax is as follows:

```
<smartcard specifier>[@<port>]@<IP address>[/<password>][#<smartcard PIN>]
```

Complete example:

```
:cs2:cjo:USB0@6071@127.0.0.1/mypwd#123456
```

- **<smartcard specifier>**

Smartcard specifiers (for example, :cs2:cjo:USB0) specify the smartcard type, the PIN pad type, and the interface (USB) to which the PIN pad is connected. See *Storage and Specification of RSA and ECDSA Keys for Authentication* in the *u.trust Anchor - csadm Manual*.

- **<port> (optional)**

If a port other than the default port 6070 has been defined in the configuration file, this port must be used here. It is the number of the port that has to be open at the PIN pad daemon on the local computer.

When connecting to a remote tools/API computer with a blocking firewall, this port also must open at the SSH daemon's side in the direction of the cHSM tool/API.

- **<IP address>**

When connecting to a remote tools/API computer without a blocking firewall, use the IP address of the local computer. When connecting to a remote tools/API computer with a blocking firewall, use 127.0.0.1 or localhost instead.

- **<password> (optional)**

This is the password for authenticating the exchange of the session key for secure messaging. This password has been defined in the password file. See [Configuring the PIN pad Daemon \(PPD\)](#) for details about the password file and the password.



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

The password must not to be confused with the smartcard PIN. The password is always entered in the command-line (preceded by a /) whereas the smartcard PIN is typically entered at the PIN pad but it can be entered in the command-line (preceded by a #) as an alternative.

The password is used for authenticating the exchange of the session key for secure messaging, and the PIN is used for authenticating the command of a cHSM tool/API (for example, csadm).

- **<smartcard PIN>** (optional)

If you do not want to enter the PIN at the PIN pad, specify it here.



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.

Use `<smartcard PIN>` only, if the cHSM tool/API expects a PIN. Otherwise, an error message is shown.

Example for a command producing an error:

```
csadm GetCardInfo=:cs2:cjo:USB0@127.0.0.1#123456
```

The following steps show two examples.

1. As a first example, the `csadm GetCardInfo` command is used.
 - a. Perform a `csadm` command according to the following examples:

```
csadm GetCardInfo=:cs2:cjo:USB0@1.1.1.1/mypwd
```

When connecting to a remote tools/API computer with a blocking firewall:

```
csadm GetCardInfo=:cs2:cjo:USB0@127.0.0.1/mypwd
```

Consider one special aspect of the `csadm GetCardInfo` command. This command only needs an installed `csadm` on the remote computer, but it does not need any contact to a cHSM.

The display of the PIN pad shows the following.

```
Insert Smartcard  
Press OK/Cancel
```

- b. Insert the smartcard into the PIN pad.
- c. Press the OK button of the PIN pad.

Example output in the command-line:

```
RSA-Key:    Utimaco IS GmbH / Init-Dev-1-Key
ECC-Key:    EcKey
RSA-Backup: KeyInfo02 #1
ECC-Backup: EcKey #1
MBK:
```

2. The second example retrieves the authentication status (permission mask) of the default administration user, ADMIN. His authentication key has been provided on every smartcard delivered by Utimaco IS GmbH.

- a. Perform a `csadm LogonSign` command according to the following examples.
When connecting to a remote tools/API computer without a blocking firewall:

```
csadm Dev=3001@194.168.4.107 LogonSign=ADMIN,:cs2:cjo:USB0@1.1.1.1
GetAuthState
```

When connecting to a remote tools/API computer with a blocking firewall:

```
csadm Dev=3001@194.168.4.107 LogonSign=ADMIN,:cs2:cjo:USB0@127.0.0.1
GetAuthState
```

The display of the PIN pad shows the following:

```
Insert Smartcard
Press OK/Cancel
```

- b. Insert the smartcard into the PIN pad.
c. Press the **OK** button of the PIN pad. Example output in the command-line:

```
current AUTH state: 22000000
user: ADMIN
```

5.3 Master Backup Key Rollover

This section is primarily intended for the cHSM Administrators and u.trust Anchor cHSM PKCS#11 users wanting to replace an existing Master Backup Key (MBK) with a new one.

u.trust Anchor cHSM Administrators can generate, import, and use a new MBK to protect external key storage as well as for creating secure backup copies of secret data and private keys stored on the u.trust Anchor cHSM from unauthorized access.

We call the process of replacing an existing MBK *Master Backup Key rollover*. This process is documented in this documentation, because some main steps are csadm commands. However, some mandatory steps must be performed by using other tools, for example, p11tool2 or P11CAT.

If you have generated backup files using the following commands/actions before the MBK rollover, these backup files are protected by the old MBK. These backup files can be restored after the MBK rollover.

- `p11tool2 BackupInternalKeys`
- P11CAT > **Backup/Restore** > **Backup/Restore Keys** > **Backup Internal Keys**
- `p11tool2 BackupExternalKeys`
- P11CAT > **Backup/Restore** > **Backup/Restore Keys** > **Backup External Keys**
- `p11tool2 BackupConfig`
- P11CAT > **Backup/Restore** > **Backup/Restore Config** > **Backup Slot Configuration Object**
- `cxitool BackupKey` (Even if this command has used a Tenant Backup Key (TBK) instead of an MBK.)
- `cngtool BackupKey`

❗ Be 100% sure that you know which MBK(s) had been used to generate these backup files (`csadm MBKListKeys` ; MBK slot 3) and that these MBKs are still available. Otherwise, these backup files are inaccessible after an MBK rollover. It is not possible to retrieve the MBK by which a backup file has been generated from this backup file.



A backup file generated by the following csadm commands is encrypted by an MBK but it is not covered by the MBK rollover. After an MBK rollover, this backup file cannot be restored and is inaccessible as long as the corresponding MBK is not imported into the MBK slot 3 in the u.trust Anchor cHSM:

- `csadm BackupDatabase`

- `csadm BackupUser`

A backup file generated by the following command is protected by other means than an MBK. An MBK rollover is irrelevant for this backup file. This backup file can be restored after an MBK rollover.

- `csadm BackupKey`

In the following descriptions, we assume that the PKCS#11 API and the u.trust Anchor cHSM are correctly configured and operational.

You will find details about how to configure u.trust Anchor cHSM PKCS#11 in the [CryptoServer – PKCS#11 p11tool2 - Reference Manual](#) provided within the product bundle under `.../Documentation/Administration Guides`.

5.3.1 MBK Rollover Preparations

Perform the following actions to prepare the MBK rollover. The execution order is irrelevant.

- Ensure that the u.trust Anchor cHSM you are going to use for the MBK rollover procedure is not currently used by any other application.
- Ensure that the tools from the product bundle are installed on the computer you will use to administer the u.trust Anchor cHSM. This guarantees that the required administration tools (`csadm`, `p11tool2`, `cxitool`, or `cngtool`) are available on the computer.
- If smartcards are used for command execution, ensure that the USB PIN pad delivered by Utimaco is connected to a USB port of the computer that `csadm` is installed on. If you want to use a PIN pad connected to another computer in the network, follow the instructions at [Using a PIN Pad](#).
- Have the smartcards with all necessary key shares of the old MBK at hand.
- Keep the number of smartcards `n` required for storing all shares of the new MBK at hand, together with the second set of `n` smartcards for the backup of the new MBK. The smartcard holders should also be present.

- Verify whether you use an internal or external key store. If the `KeysExternal` parameter in the `cs_pkcs11_r2.cfg` configuration file has been set to true, an external key store is used. Consequently, the `KeyStore` parameter in the same file must have been set as well. Verify this by following the instructions in *Editing the cs_pkcs11_R2.cfg Configuration File* in the [CryptoServer – PKCS#11 P11CAT Manual](#). If the `KeysExternal` parameter is set to `false`, an internal key store is used.
- A user with at least permission 2 in the user group 6 (02000000) should be available for authenticating the generation of the new MBK. Alternatively, if authentication according to the two-person rule is used, at least two users, each with permission 1 in the user group 6 (01000000) should be available.
- A user with key manager permissions should be available. The key manager permission mask is configured by the `CKA_CFG_AUTH_KEYM_MASK` parameter. Verify this by following the instructions in *GetGlobalConfig* or *GetSlotConfig* in the [CryptoServer – PKCS#11 P11CAT Manual](#). If the `KeysExternal` parameter is set to false, an internal key store is used.
- A user with at least permission 2 in the user groups 6 and 7 (22000000) should be available for authenticating the backup procedure of the active MBK. Alternatively, if authentication according to the two-person rule is used, at least two users, each with permission 1 in the user groups 6 and 7 (11000000) should be available.
- If you are using a u.trust Anchor cHSM in Cluster Mode, the same MBK should be available on all devices in the cluster. In this case, ensure that the same active MBK has been imported on all u.trust Anchor cHSMs used to generate and store cryptographic keys into the (the same) external key storage.

5.3.2 MBK Rollover of an Internal Keystore



Pay attention to the warnings in the [Master Backup Key Rollover](#) chapter.

Procedure

1. Follow the instructions in [MBK Rollover Preparations](#).
2. Back up all cryptographic keys of a PKCS#11 slot in the internal key store by following either the instructions in section *BackupInternalKeys* in the [CryptoServer – PKCS#11 p11tool2 Reference Manual](#) or in the section *Creating a Backup of all Keys in a Slot* in

the [CryptoServer – PKCS#11 P11CAT Manual](#).

Example:

```
p11tool2 Slot=0 Login=KeyMgr,123456 BackupInternalKeys=C:\Utlimaco\CryptoServer\Administration\internal_keys_p11slot0.bak
```

Example output:

```
2 internal key(s) backed up
```

3. Repeat this step for all other PKCS#11 slots.
4. Verify which MBK is stored in the cHSM with the `csadm MBKListKeys` command.

Example output:

slot	name	len	algo	type	k	generation	date	key	check	value
3	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A	6228784FDFB0DF27	

A current (old) AES MBK is stored in MBK slot 3. MBK slots numbers must not be confused with PKCS#11 slot numbers.

⚠ Ensure that the value shown in the `algo` column is AES. Otherwise, do not continue the MBK rollover.

5. Verify that key shares of this old MBK are stored, for example, on the smartcards. You can perform the `csadm MBKCardInfo` command to do so.

Example output for the first smartcard:

REC	TYPE	ALG	LEN	NAME	TIME	GEN	K	ID	HASH
15	XOR	AES	256	MbkAes30	09.05.2019	08:57:24	0	0	D0779FB705960D8A

The key shares of one MBK are usually stored in the same record number on the smartcards (15 in the example). 15 is the default value for AES MBKs.

Use these smartcards to import the MBK in the next step.

- Import the old MBK to a MBK slot number greater than 3 into the cHSM, for example by using the MBK shares on the smartcards. Use the `csadm MBKImportKey` command.

Example:

```
csadm Dev=3001@194.168.4.107 LogonPass=God,123456 Key=:cs2:cjo:USB0,15
MBKImportKey=7
```

Follow the instructions on the PIN pad display.

- Verify that the old MBK now is available in two MBK slots in the cHSM. You can perform the `csadm MBKListKeys` command to do so.

Example output:

slot	name	len	algo	type	k	generation	date	key	check	value
3	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A:	6228784FDFB0DF27	
7	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A:	6228784FDFB0DF27	

- If no new MBK is available yet, generate a new MBK for the cHSM by using the `csadm MBKGenerateKey` command. For the following example you have to keep two smartcards at hand for the MBK generation as well as the smartcards of both users who have to authenticate the command.

Example:

```
csadm Dev=3001@194.168.4.107 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,32,2,2,MbkAes29
```

Follow the instructions on the PIN pad display to authenticate the `csadm MBKGenerateKey` command, and then to generate the MBK.

- If not already done yet, change the PIN of all smartcards whereon a share of the MBK is stored by using the `csadm MBKPINChange`.

Example: The two generated MBK shares are stored on two smartcards, so the following command must be repeated for both smartcards.

```
csadm MBKPINChange=:cs2:cjo:USB0
```

10. Follow the instructions on the PIN pad display.
11. Import the new MBK into the cHSM by using the `csadm MBKImportKey` command. It is important to import it to MBK slot 3.

Example:

12. Execute the `csadm MBKListKeys` command to ensure that the MBK has been successfully imported into the cHSM.

Example:

```
csadm Dev=3001@194.168.4.107 MBKListKeys
```

Example output:

slot	name	len	algo	type	k	generation date	key check value
3	MbkAes29	32	AES	XOR	2	2019/05/09 09:04:11	2722BDA0D7D6E821:8F21825F743C3A49
7	MbkAes30	32	AES	XOR	2	2019/05/09 08:57:24	D0779FB705960D8A:6228784FDFB0DF27

13. Consider the following warning.

 The backup file(s) created in step 3 and 4 on page 293 can only be used to restore the internal PKCS#11 key store if the old MBK is still present in the cHSM (MBK slot 7 in the example). Do not overwrite the MBK in this MBK slot by importing an MBK into this slot unless you have a backup for this MBK; for example, MBK shares on smartcards.

14. Now, the MBK rollover is finished. You can verify the result of the MBK rollover by performing the following steps.
 - a. Delete a cryptographic key in the internal key store that you do not need anymore by performing the `p11tool2 DeleteObject` command, see section *DeleteObject* in the [CryptoServer – PKCS#11 p11tool2 Reference Manual](#), or by following the instructions in *Deleting Objects from the CryptoServer* in the [CryptoServer –](#)

[PKCS#11 P11CAT Manual](#) for this key.

Example:

```
p11tool2 Slot=0 LoginUser=123456 Label="RSA Public Key" Id="P12"  
DeleteObject
```

Example output:

```
1 Objects deleted
```

- b. Note down the slot number of the PKCS#11 slot the cryptographic key has been deleted from.
- c. Restore the cryptographic keys of the corresponding slot in the internal PKCS#11 key store by performing a `p11tool2 RestoreInternalKeys` command, see section *RestoreInternalKeys* in the [CryptoServer – PKCS#11 p11tool2 Reference Manual](#) or by following the instructions in *Restoring a Backup of All Keys in a Slot* in the [CryptoServer – PKCS#11 P11CAT Manual](#). Use the PKCS#11 slot number and the backup file that have been specified in the previous steps.

Example:

```
p11tool2 Slot=0 Login=KeyMgr,123456 RestoreInternalKeys=C:  
\\Utimaco\\CryptoServer\\Administration\\internal_keys_p11slot0.bak
```

Example output:

```
2 internal keys restored to internal key store
```

There is no need to specify the MBK slot number of the old MBK (7 in the example) in the `csadm` command, but this MBK slot number is not shown in the output either. The CHSM automatically finds the correct MBK slot number.

- d. Verify whether the cryptographic key that you have deleted in step 14a) has been restored. To do so, follow either the instructions in `ListObjects` in the [CryptoServer – PKCS#11 p11tool2 Reference Manual](#) or the first steps in section *Generating a Key* (only up to and including the step *Click Object Management*) in [CryptoServer – PKCS#11 P11CAT Manual](#) for the corresponding PKCS#11 slot.

5.3.3 MBK Rollover of an External Keystore



Pay attention to the warnings in the [Master Backup Key Rollover](#) chapter.

Procedure

1. Follow the instructions in [MBK Rollover Preparations](#).
2. Verify which MBK is stored in the cHSM. You can perform the `csadm MBKListKeys` command to do so.

Example output:

```
slot name      len algo type   k generation date      key check value
-----
3      MbkAes30 32 AES  XOR    2  2019/05/09 08:57:24
D0779FB705960D8A:6228784FDFB0DF27
```

A current (old) AES MBK is stored in MBK slot 3. MBK slots numbers must not be confused with PKCS#11 slot numbers.

3. Verify that key shares of this old MBK are stored, for example, on the smartcards. You can perform the `csadm MBKCardInfo` command to do so.

Example output for the first smartcard:

```
REC TYPE  ALG LEN NAME      TIMEGEN              K  ID HASH
-----
15  XOR   AES 256 MbkAes30 09.05.2019 08:57:24 0 0  D0779FB705960D8A
```

In the example, an XOR type is shown. That means that only two key shares are needed. Example output for the second smartcard:

```
REC TYPE  ALG LEN NAME      TIMEGEN              K  ID HASH
-----
15  XOR   AES 256 MbkAes30 09.05.2019 08:57:24 0 0  6228784FDFB0DF27
```

The key shares of one MBK are usually stored in the same record number on the smartcards (15 in the example). 15 is the default value for AES MBKs.

Use these smartcards to import the MBK in the next step.

4. Import the old MBK to an MBK slot number greater than 3 into the cHSM for example, by using the MBK shares on the smartcards. Use the `csadm MBKImportKey` command.

Example:

```
csadm Dev=3001@194.168.4.107 LogonPass=God,123456 Key=:cs2:cjo:USB0,15
MBKImportKey=7
```

Follow the instructions on the PIN pad display.

5. . Verify which MBK is stored on the device by performing the `csadm MBKListKeys` command.

slot	name	len	algo	type	k	generation	date	key	checkvalue
3	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A:6228784FDFB0DF27	
7	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A:6228784FDFB0DF27	



Ensure that the value shown in the `algo` column is AES. Otherwise, do not continue the MBK rollover.



A current (old) AES MBK is stored in MBK slot 3. MBK slots numbers must not be confused with PKCS#11 slot numbers

6. If no new MBK is available yet, generate a new MBK for the cHSM by using the `csadm MBKGenerateKey` command. For the following example, you have to keep two smartcards at hand for the MBK generation as well as the smartcards of both users who have to authenticate the command.

Example:

```
csadm Dev=3001@194.168.4.107 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,32,2,2,MbkAes29
```

Follow the instructions on the PIN pad display to authenticate the csadm `MBKGenerateKey` command, and then to generate the MBK.

7. If not already done yet, change the PIN of all smartcards whereon a share of the MBK is stored by using the `csadm MBKPINChange` command.

Example:

The two generated MBK shares are stored on two smartcards, so the following command must be repeated for both smartcards.

```
csadm MBKPINChange=:cs2:cjo:USB0
```

8. Follow the instructions on the PIN pad display.
9. Import the new MBK into the cHSM by using the `csadm MBKImportKey` command. It is important to import it to MBK slot 3.

Example:

```
csadm Dev=3001@194.168.4.107 LogonSign=Admin1,:cs2:cjo:USB0
LogonSign=Admin3,:cs2:cjo:USB0 Key=:cs2:cjo:USB0,14,:cs2:cjo:USB0,14
MBKImportKey=3
```

10. Execute the `csadm MBKListKeys` command to ensure that the MBK has been successfully imported into the cHSM.

Example:

```
csadm Dev=3001@194.168.4.107 MBKListKeys
```

Example output:

slot	name	len	algo	type	k	generation	date	key	checkvalue
3	MbkAes29	32	AES	XOR	2	2019/05/09	09:04:11	2722BDA0D7D6E821:8F21825F743C3A49	
7	MbkAes30	32	AES	XOR	2	2019/05/09	08:57:24	D0779FB705960D8A:6228784FDFB0DF27	

11. Perform the `p11tool2 RecryptExternalKeys` command. This command first backs up all cryptographic keys of the specified PKCS#11 slot of the external PKCS#11 key store and stores them in the specified backup file. Then it encrypts the cryptographic keys in this PKCS#11 slot of the external PKCS#11 key store with the new MBK, see `RecryptExternalKeys` in the [CryptoServer - PKCS#11 p11tool2 - Reference Manual](#). This step cannot be performed by using P11CAT.


Example:

```
p11tool2
Slot=0 Login=USR_0000,123456 RecryptExternalKeys=C:
\Utimaco\CryptoServer\Administration\p11.pks.bak
```

Example Output:

```
2 external key(s) backed up
2 external key(s) recrypted
```

12. Repeat step 11 for all other PKCS#11 slots of the external PKCS#11 keystore containing cryptographic keys.

 The backup file(s) can only be used to restore the external PKCS#11 key store if the old MBK is still present in the cHSM (MBK slot 7 in the example). Do not overwrite the MBK in this MBK slot by importing an MBK into this slot unless you have a backup for this MBK, for example, MBK shares on smartcards.

13. Now, the MBK rollover is finished. You can verify the result of the MBK rollover by performing the following steps.
 - a. Delete a cryptographic key in the external key store that you do not need anymore by performing the `p11tool2 DeleteObject` command, see `DeleteObject` in the [CryptoServer - PKCS#11 p11tool2 - Reference Manual](#), or by following the instructions in *Deleting Objects from the CryptoServer* in the [CryptoServer – PKCS#11 P11CAT Manual](#) for this key.

Example:

```
p11tool2 Slot=0 LoginUser=123456 Label="RSA Public Key" Id="P12"
DeleteObject
```

Example Output:

1 Objects deleted

- b. Note down the slot number of the PKCS#11 slot from which the cryptographic key has been deleted.
- c. Restore the cryptographic keys of the corresponding slot in the external PKCS#11 key store by performing a `p11tool2 RestoreExternalKeys` command, see *RestoreExternalKeys* in the [CryptoServer - PKCS#11 p11tool2 - Reference Manual](#) or by following the instructions in *Restoring a Backup of All Keys in a Slot* in the [CryptoServer – PKCS#11 P11CAT Manual](#). Use the PKCS#11 slot number and the backup specified in the previous steps.

Example:

```
p11tool2Slot=0 Login=KeyMgr,123456
RestoreExternalKeys=C:
\Utimaco\CryptoServer\Administration\external_keys_p11slot0.bak
```

Example output:

2 external keys restored to external key store

There is no need to specify the MBK slot number of the old MBK (7 in the example) in the `csadm` command but this MBK slot number is also not shown in the output. The cHSM automatically finds the correct MBK slot number.

- d. Verify whether the cryptographic key that you have deleted in step 13a) has been restored. To do so, follow either the instructions in section *ListObjects* in the [CryptoServer - PKCS#11 p11tool2 - Reference Manual](#) or the first steps in section *Generating a Key* (only up to and including the step *Click Object Management*) in the [CryptoServer – PKCS#11 P11CAT Manual](#) for the corresponding PKCS#11 slot.

5.4 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 *Using the Signed Configuration File cmds.scf* in the [u.trust Anchor - csadm Manual](#) for step-by-step instructions on how to create/edit, sign, load and activate your signed configuration file `cmds.scf`.

6 Monitoring and Maintenance

6.1 Indicating a FIPS cHSM

To indicate whether a cHSM is a FIPS cHSM, perform the `csadm GetState` command.

<i>Return</i>	<i>Description</i>
FIPS mode	ON: The u.trust Anchor device is using the FIPS firmware image and cHSM is using the SecurityServer FIPS template. This parameter is not returned, if the u.trust Anchor device is using a non-FIPS firmware image.
FIPS restrictions	applied: The cHSM is using the SecurityServer FIPS template and u.trust Anchor device is using a non-FIPS firmware image. This parameter is not returned, if the cHSM is using a non-FIPS template.

6.2 Leaving the FIPS Error State

In this section it will be explained how to leave a the state ERROR, see [cHSM Operating States](#).

Precondition

The cHSM is in a FIPS ERROR state, which is determined by the `csadm GetState` command returning the parameter `state = INITIALIZED ERROR`.

Leaving an Error State

In the case that a FIPS cHSM is in ERROR state, perform a restart of the cHSM with the `csadm Restart` command. If this does not resolve the ERROR state, please contact the Global Administrator.

6.3 Logs

Every action that involves a security issue, such as creating a new user or an event, is saved in logs. There are two types of logs: the boot log and the audit log.

6.3.1 Boot Log

Every time the cHSM starts up, every individual step and its results are recorded in the boot log. These include:

- The version number of the loaded operating system (SMOS)

- The version number of the internal sensor switch
- Information about if a license file is present and which system settings have been made on the basis of this license file
- For every firmware module that was loaded and started by the operating system: a short description and unique ID to identify it, whether it could be initialized successfully or which errors occurred during the initialization process.

If any problems occur during the boot process, the boot can be used log to analyze the errors. The boot log is stored as a text file on the cHSM.

6.3.2 Audit Log

The cHSM offers extensive audit functionality. Some events will always and automatically be audited. Other events will optionally be audited if the cHSM is accordingly configured.

- A timestamp with date and time of the event (if the RTC is running),
- User name of all users who authenticated the audited command (if appropriate),
- Function code (FC) and sub-function 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.

Every entry in the audit log has the following structure:

DD.MM.YY hh:mm:ss [user] event data [returncode] <CR-LF>

The individual fields in this type of entry have the following meaning: @mlr

Field	Description
DD.MM.YY	Date on which the event occurred in <code>day.month.year</code> format. The date is taken from the cHSM's internal real time clock.
hh:mm:ss	Time at which the event occurred in <code>hour:minute:second</code> format. The time is taken from the cHSM's internal real time clock.
[user]	Name(s) of the user(s) who performed the action. Multiple users are separated by commas. However, if an "external" event occurs, i.e. alarm or restart, the user ID is missing.

Field	Description
event	Description of the event in a readable format (see table below for a list of all auditable events in FIPS cHSMs).
data	Additional information about the event in a readable format
[returncode]	Return value of the function that has been performed. A return value of 0 means that the function has been performed successfully. A return code other than 0 shows the returned error number.
<CR-LF>	Carriage return - line feed

Table 35: Meaning of the audit log entry fields

The square brackets [...] displayed for user and return code in the previous table do not mean that these fields are optional. In each case the relevant user name or return code is effectively enclosed in square brackets.

For details, see [Audit Log Entries](#).

6.3.2.1 Audit Log Entries

The following table provides an overview of the used audit message classes.

Audit message class name	Audit message class mask	Audit message class number	Description / Event name	Default
OS_AUDIT_CLASS_FIRMWARE	0x00000001	1	Firmware Management	Yes
OS_AUDIT_CLASS_USER	0x00000002	2	User Management	Yes
OS_AUDIT_CLASS_TIME	0x00000004	3	Date/Time Management	Yes
OS_AUDIT_CLASS_STARTUP	0x00000008	4	Startup Messages	Yes
OS_AUDIT_CLASS_AUDIT	0x00000010	5	Audit Log Management	Yes
OS_AUDIT_CLASS_MBK	0x00000020	6	MBK Management	Yes
OS_AUDIT_CLASS_KEY	0x00000040	7	Key Management	No
OS_AUDIT_CLASS_AUTH_SUCCESS	0x00000080	8	Successful Login Attempts	No
OS_AUDIT_CLASS_AUTH_FAILED	0x00000100	9	Failed Login Attempts	Yes

<i>Audit message class name</i>	<i>Audit message class mask</i>	<i>Audit message class number</i>	<i>Description / Event name</i>	<i>Default</i>
OS_AUDIT_CLASS_BACKUP_RESTORE	0x00000200	10	Backup/Restore	Yes
OS_AUDIT_CLASS_SYSTEM	0x00000400	11	Operating System Events	Yes
OS_AUDIT_CLASS_ACTION_NEEDED	0x00000800	12	Action needed	Yes
<Class reserved for future use>	-	13-23,31	-	
OS_AUDIT_CLASS_CUSTOM_MASK	-	24-30	-	
OS_AUDIT_CLASS_ALWAYS	0xFFFFFFFF	-	-	
<Special audit message class>	0x00000001	-	-	

Table 36: Audit message classes

The first column shows the name of the audit message class. The second and the third columns show the audit message class mask and number as used in the `csadm GetAuditConfig` command and in the `csadm SetAuditConfig` command.

the `csadm GetAuditConfig` command and in the `csadm SetAuditConfig` command.

Examples:

```
csadm dev=192.168.4.1 getauditconfig
Audit log configuration parameters:
Number of logfiles: 3
Rotate logfiles: yes
Max filesize: 200000
Events: 0x00000010 (Bits 5)
```

```
csadm Dev=192.168.4.1 LogonPass=sven,swordfish
SetAuditConfig=MaxFileCount=5,Events=1:2:3:4:5:6:8:9
csadm Dev=192.168.4.1 LogonSign=nils,:cs2:cjo:USB0
SetAuditConfig=RotateLogfiles=yes,Events=0x000001BF
```

In the examples, 0x00000010 and 0x000001BF are the masks, and 5 and 1:2:3:4:5:6:8:9 are the corresponding numbers.

The fourth column shows the event name. The fifth column shows whether the audit message class is enabled by default.

The following table shows the function codes for the firmware modules:

Function Code	Firmware Module
0x000	SMOS
0x068	CXI
0x069	MBK, extended interface
0x083	CMDS
0x087	ADM
0x088	DB
0x096	MBK, normal interface

Table 37: Table 35: Function codes and firmware modules

Each firmware module specification contains more detailed information about its audit log entries.

The following table shows which function codes including the subfunction code and subfunction name belong to a dedicated audit message class. If a function code cannot be assigned to an audit message class, it is not shown in this table.

Audit Message Class Name	Function code, subfunction code and subfunction information	Comment
OS_AUDIT_CLASS_FIRMWARE	FC:0x083 SFC:0x13 SetAdminMode	Neither the function code nor the subfunction code is shown in the audit log entry
	FC:0x083 SFC:0x14 SetStartupMode	
	FC:0x087 SFC:0x03 LoadFile	
	FC:0x087 SFC:0x04 DeleteFile	
	FC:0x087 SFC:0x13 LoadFirmwareDecryptionKey	
OS_AUDIT_CLASS_USER	FC:0x068 SFC:0x21 AddUser	This is the CXI AddUser subfunction, not the CMDS AddUserExtended subfunction
	FC:0x068 SFC:0x22 DeleteUser	This is the CXI DeleteUser subfunction
	FC:0x083 SFC:0x05 DeleteUser	This is the CMDS DeleteUser subfunction

	FC:0x083 SFC:0x06 ChangeUser	
	FC:0x083 SFC:0x0D RestoreUser	This audit log entry is created if either OS_AUDIT_CLASS_BACKUP_RESTORE or OS_AUDIT_CLASS_USER is enabled.
	FC:0x083 SFC:0x0E AddUserExtended	This is the CMDS AddUserExtended subfunction, not the CXI AddUser subfunction
	FC:0x083 SFC:0x11 SetMaxAuthFailures	Neither the function code nor the subfunction code is shown in the audit log entry.
OS_AUDIT_CLASS_STARTUP	FC:0x... SFC:0x... Checking configured authentication requirements in the audit log	
	FC: 0x000 SCF: ... SMOS successfully started	
OS_AUDIT_CLASS_AUDIT	FC:0x087 SFC:0x0B ClearAuditLog	
	FC:0x087 SFC:0x1D GenerateAuditLogKey	
	FC:0x087 SFC:0x20 ClearAuditLogFiles	
OS_AUDIT_CLASS_MBK	FC:0x069 SFC:0x00 Import DES key from PIN pad16	This is the extended MBK interface with the function code 0x069, not the normal MBK interface with the function code 0x096. Neither the function code, nor the subfunction code is shown in the audit log entry.
	FC:0x069 SFC:0x01 Import DES key from smartcard16	
	FC:0x069 SFC:0x02 Import AES key from smartcard16	
	FC:0x069 SFC:0x03 Generate DES key and write to smartcard16	
	FC:0x069 SFC:0x04 Generate AES key and write to smartcard16	
	FC:0x069 SFC:0x08 Import MBK key from PIN pad and store on smartcard16	
	FC:0x069 SFC:0x09 Generate DES key shares and store on smartcard16	
	FC:0x069 SFC:0x0A Generate AES key shares on store on smartcard16	
	FC:0x096 SFC:0x04 GenerateMBK	

OS_AUDIT_CLASS_KEY	FC:0x096 SFC:0x05 ImportMBK	
	FC:0x068 SFC:0x05 InitKeyGroup	
	FC:0x068 SFC:0x08 BackupKey	
	FC:0x068 SFC:0x09 RestoreKey	
	FC:0x068 SFC:0x0B GenerateKey	
	FC:0x068 SFC:0x0C OpenKey	OpenKey only creates audit log entries for SecurityServer/ CryptoServer SDK versions earlier than 4.30 and CXI firmware module versions earlier than 2.4.0.0
	FC:0x068 SFC:0x0D DeleteKey	
	FC:0x068 SFC:0x0F SetKeyProp	
	FC:0x068 SFC:0x10 ExportKey	
	FC:0x068 SFC:0x11 ImportKey	
	FC:0x068 SFC:0x19 CreateObject	
	FC:0x068 SFC:0x1A GenerateKeyPair	
	FC:0x068 SFC:0x1B CopyObject	
	FC:0x068 SFC:0x1C DeriveKey	
	FC:0x068 SFC:0x1D WrapKey	
	FC:0x068 SFC:0x1E UnwrapKey	
	FC:0x068 SFC:0x2B SplitKey	
OS_AUDIT_CLASS_AUTH_SUCCESS	-	
OS_AUDIT_CLASS_AUTH_FAILED	-	
OS_AUDIT_CLASS_BACKUP_RESTORE	FC:0x083 SFC:0x0C BackupUser	
	FC:0x083 SFC:0x0D RestoreUser	This audit log entry is created if either OS_AUDIT_CLASS_BACKUP_RESTORE or OS_AUDIT_CLASS_USER is enabled.
	FC:0x087 SFC:0x18 ExportDBEntry	
	FC:0x087 SFC:0x19 ImportDBEntry	
OS_AUDIT_CLASS_SYSTEM	FC: 0x000 SCF: ... CryptoServer shutdown	
	FC: 0x000 SCF: ... SHA-512 known answer test failed	
	FC: 0x000 SCF: ...Unpacking firmware module failed	
	FC: 0x000 SCF: ... Loading firmware module failed	

	FC: 0x000 SCF: ... Start function of firmware module failed	
	FC: 0x000 SCF: ... Firmware module is defective	
	FC: 0x000 SCF: ... DRBG: Known answer test failed	
	FC: 0x000 SCF: ... DRBG: Initialization failed	
	FC: 0x000 SCF: ... DRBG: Random number generation failed	
	FC: 0x000 SCF: ... DRBG: Reseed failed	
	FC: 0x000 SCF: ... TRNG: Hardware failed	
	FC: 0x000 SCF: ... TRNG: Self-test failed	
	FC: 0x000 SCF: ... TRNG: Online test failed	
	FC: 0x088 SFC: Corrupted database	
OS_AUDIT_CLASS_ACTION_NEEDED	-	
<Class reserved for future use>	-	
OS_AUDIT_CLASS_CUSTOM_MASK	-	
OS_AUDIT_CLASS_ALWAYS	FC: 0x000 SCF: ... FIPS mode entered or left	
	FC: 0x000 SCF: ... Clear to factory settings	
	FC: 0x000 SCF: ... Erase executed	
	FC: 0x000 SCF: ... New alarm detected	
	FC: 0x000 SCF: ... Temperature exceeds valid range	
	FC: 0x087 SFC: 0x15 Clear	
	FC: 0x087 SFC: 0x1A ConfigParamSet	
<Special audit message class>	FC: 0x087 SFC: 0x14 ResetAlarm	

Table 38: Function codes and subfunction codes of audit messages

The tables in the following chapters contain all possible audit log entries. They also contain information by which of the following tools and actions an audit log entry is generated:

- csadm

- PKCS#11 CryptoServer Administration Tool (p11CAT)
- p11tool2
- cxitool
- cngtool

Note that the API calls resulting in audit log entries are not listed.

The general format of an audit message is <DateTime> <User> <Function code> <Subfunction code> <Log data>.

6.3.2.1.1 OS_AUDIT_CLASS_FIRMWARE

The OS_AUDIT_CLASS_FIRMWARE audit message class is mainly used when firmware modules are loaded, deleted etc. The following tables describe the corresponding audit log entries and the actions leading to them.

<i>Audit log entry</i>	<i>Description</i>
SetAdminMode from <old> to <new> [error code]	Setting the administration mode Neither the function code (0x83) nor the subfunction code (0x13) is shown in the audit log entry. Initiation: csadm SetAdminMode
SetStartupMode <new> [error code]	Setting the startup mode Neither the function code (0x83) nor the subfunction code (0x14) is shown in the audit log entry. Initiation: csadm SetStartupMode
FC:0x087 SFC:0x03 Load File '<filename>' Part 1 [error code]	Loading the <filename> file. The part of the file is optional. Initiation: csadm LoadFile or csadm LoadPKG or CAT (Manage > Firmware > Update (installs only new firmware) or Manage > Firmware > New installation (deletes all files; installs new firmware package))
FC:0x087 SFC:0x03 Verification of MMC signature failed (<filename>) err = <error code>	The <filename> file (.mtc file) is a firmware module and the verification of its MMC signature has failed with the error <error code>. Initiation: csadm LoadFile
FC:0x087 SFC:0x04 Delete File '<filename>' [error code]	Deleting the <filename> file. Initiation: csadm DeleteFile or csadm LoadPKG with option ForceClear or CAT (Manage > Firmware > New installation (deletes all files; installs new firmware package))

Audit log entry	Description
FC:0x087 SFC:0x13 Load FW DecKey #R0 [error code]	Loading the firmware description key. Initiation: csadm LoadFWDecKey or CAT (Manage > Firmware Decryption Key)

Table 39: Audit log entries for OS_AUDIT_CLASS_FIRMWARE

6.3.2.1.2 OS_AUDIT_CLASS_USER

The OS_AUDIT_CLASS_USER audit message class is used when users are created, changed, restored and deleted. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
FC:0x068 SFC:0x21 user_add: <key group> <user name> (<Optional user name>, <authentication mechanism ID> <user's permission>) [error code]	Adding a user in CXI Initiation: P11CAT (e.g., Slot Management > Init Token > SO PIN, Confirm SO PIN > Init Token > Login)
FC:0x068 SFC:0x22 user_delete: <key group> <user name> [error code]	Deleting a user in CXI Initiation: P11CAT (Login/Logout > Login Generic > Login > Slot Management > Delete SO > Delete SO > Yes)
FC:0x083 SFC:0x05 Delete User '<user name>' [error code]	Deleting a user in CMDS Initiation: csadm DeleteUser or CAT (Manage > User > Delete User)
FC:0x083 SFC:0x06 Change User '<user name>' [error code]	Changing a user. Initiation: csadm ChangeUser CAT (Manage > User > Change Token/Password)
FC:0x083 SFC:0x0D Restore User '<user name> (<Optional user name>, <authentication mechanism ID> <user's permission>) [error code]	Restoring a user. Initiation: csadm RestoreUser or CAT (Manage > Manage User > Restore Users) This audit log entry is created if either OS_AUDIT_CLASS_BACKUP_RESTORE or OS_AUDIT_CLASS_USER is enabled.

Audit log entry	Description
FC:0x083 SFC:0x0E Add User '<user name>' (<Optional user name>, <authentication mechanism ID> <user's permission>) [error code]	Adding a user in CMDS Initiation: csadm AddUser or CAT (Manage > User > Add User)
Set MaxAuthFailures from <old> to <new> [error code]	Setting the maximum number of authentication failures Neither the function code (FC: 0x083) nor the subfunction code (SFC: 0x11) is shown in the audit log entry. Initiation: csadm SetMaxAuthFails

Table 40: Audit log entries for OS_AUDIT_CLASS_USER

6.3.2.1.3 OS_AUDIT_CLASS_TIME

The OS_AUDIT_CLASS_TIME audit message class is used when the time is set, the limitations for time changes changed and the activation state for these changes is changed. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
FC:0x087 SFC:0x07 Set Time from <date/time stamp> [error code]	Setting the time Initiation: csadm SetTime or CAT (Manage > Date/Time)
FC:0x09A SFC:0x02 NTP: New Settings MaxAdjustPerOperation=<value1> MaxAdjustPerDay=<value2> [error code]	Changing the activation state Initiation: CAT (Manage > NTP Settings > Enabled / Disabled)
FC:0x9A7 SFC:0x04 NTP: New Settings MaxAdjustPerOperation=<value1> MaxAdjustPerDay=<value2> [error code]	Setting the NTP settings Initiation: CAT (Manage > NTP Settings > Max. time to set per day and Manage > NTP Settings > Max. time to set per operation)

Table 41: Audit log entries for OS_AUDIT_CLASS_TIME

6.3.2.1.4 OS_AUDIT_CLASS_STARTUP

The OS_AUDIT_CLASS_STARTUP audit message class is used when the configured authentication requirements in the audit log are checked the power-on self-test has failed, the

cryptographic algorithm test has failed, SMOS is started etc. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
FC:<function code> SFC: <subfunction code> Configured Permission=<value>	Checking the configure authentication requirements in the audit log Initiation: <code>csadm - Edit the <code>cmds.scf</code> file, perform <code>csadm SignConfig</code>, <code>csadm LoadFile=...\cmds.scf</code>, <code>csadm ListFiles</code>, <code>csadm Restart</code>, <code>csadm GetBootLog</code> and <code>csadm GetAuditLog</code></code> For details, see <i>Using the Signed Configuration File cmds.scf</i> in the <i>u.trust Anchor - csadm Manual</i> and <i>Creating and Using the Signed Configuration File cmds.scf</i> in the <i>CryptoServer – csadm Manual</i> .
CMDS: unable to initialize module POST for Power-on Self-tests (err = <error code>)	Failed power-on self-test initialization. Initiation: The POST (power-on self-test) firmware module is not available and one of the following actions is performed: Windows command-line (shutdown -r -t 0), <code>csadm Restart</code> or <code>csadm CSLReboot</code> or CAT: Manage > Reboot CryptoServer
POST: Selftest of utility functions failed with error code: <error code>	Failed power-on self-test
POST: <Algorithm name> Cryptographic Test failed with error code: <error code>	Failed cryptographic algorithm test, i.e., for AES, ECDA, HASH, HMAC RSA, DSA or DES
POST: <Algorithm name> Cryptographic Test skipped	Skipped cryptographic algorithm test, i.e., for AES, ECDA, HASH, HMAC RSA, DSA or DES Initiation: The corresponding firmware module (AES, ECDSA etc.) is not available and one of the following actions is performed: Windows command-line (shutdown -r -t 0), <code>csadm Restart</code> or <code>csadm CSLReboot</code> . Or, CAT: Manage > Reboot CryptoServer
SMOS Ver. <version> successfully started	SMOS has been started successfully Initiation: <code>csadm Restart</code> or CAT: Manage > Reboot CryptoServer

Table 42: Audit log entries for OS_AUDIT_CLASS_STARTUP

6.3.2.1.5 OS_AUDIT_CLASS_AUDIT

The OS_AUDIT_CLASS_AUDIT audit message class is used when the audit log configuration is cleared or changed. The following tables describe the corresponding audit log entries and the actions leading to them.

<i>Audit log entry</i>	<i>Description</i>
FC:0x087 SFC:0x0B Clear Audit Files '<filename>' (<number>) [<error code>]	Deleting audit log file(s) <number> indicates the number of the (youngest) parts that are not deleted (omitted in error case). Initiation: csadm ClearAuditLog or CAT: Manage > Audit Log > Clear Log > Yes . These actions can be initiated in the Operational Mode and in the Maintenance Mode. See below for another method to delete audit log files (SFC = 0x20).
Successful execution: FC:0x087 SFC:0x1D Generate Audit Log Key with mode <mode> and public key fingerprint <fingerprint> Failed execution: FC:0x087 SFC:0x1D Generate Audit Log Key ERROR [<error code>]	Generating an audit log signature key <mode> indicates the mode of the audit log signature key (see description of the <mode> parameter in section <i>GenerateAuditLogKey</i> in u.trust Anchor - csadm Manual for details) and <fingerprint> indicates the public key fingerprint. Initiation: csadm GenerateAuditLogKey
Successful execution: FC:0x087 SFC:0x20 Clear Audit Files (audit<first>.log - audit<last>.log) Failed execution: FC:0x087 SFC:0x20 Clear Audit Files ERROR [<error code>]	Deleting audit log file(s) <first> indicates the first audit log file that has been deleted, and <last> indicates the last file that has been deleted with both of them being a 6-digit hexadecimal number without a leading 0x. Initiation: csadm ClearAuditLogFiles See above for another method to delete audit log files (SFC = 0x0B).

Table 43: Audit log entries for OS_AUDIT_CLASS_AUDIT

6.3.2.1.6 OS_AUDIT_CLASS_MBK

The OS_AUDIT_CLASS_MBK audit message class is used when an MBK (master backup key) is generated, imported and stored. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
mbk_ei_import_pp: DES-16, slot <slot number> [error code]	Importing a DES key from the PIN pad. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x00) is shown in this audit log entry.
mbk_ei_import_des_sc: DES- 16, '<key name>', slot <slot number> [error code]	Importing a DES key from the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x01) is shown in this audit log entry.
mbk_ei_import_aes_sc: AES- 32, '<key name>', slot <slot number> [error code]	Importing an AES key from the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x02) is shown in this audit log entry.
mbk_ei_generate_des_sc: DES-16, '<key name>', record <record number> [error code]	Generating a DES key and writing to the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x03) is shown in this audit log entry.
mbk_ei_generate_aes_sc: AES-32, '<key name>', record <record number> [error code]	Generating an AES key and writing to the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x04) is shown in this audit log entry.
mbk_ei_imp_pp_write_sc: <key info>, '<key name>', record <record number> [error code]	Importing an MBK key from the PIN pad and storing them on the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x08) is shown in this audit log entry.
mbk_ei_gen_des_key_shares: DES-16 '<key name>', <k>- out-of-<n>, record <record number> [error code]	Generating DES key shares and storing them on the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x09) is shown in this audit log entry
mbk_ei_gen_aes_key_shares: AES-32 '<key name>', <k>- out-of-<n>, record <record number> [error code]	Generating AES key shares and storing them on the smartcard. Neither the function code (FC:0x069) nor the subfunction code (SFC:0x0A) is shown in this audit log entry.
mbk_key_generate: <key info> '<key name>' <k>- out-of-<n> [error code]	Generating an MBK. Initiation: <code>csadm MBKGenerateKey</code> or CAT: Manage > Master Backup Key > Generate
mbk_key_import: <key info> '<key name>', <x> parts, slot <slot number> [error code]	Importing an MBK. Initiation: <code>csadm MBKImportKey</code> or CAT: Manage > Master Backup Key > Import

Table 44: Audit log entries for S_AUDIT_CLASS_MBK

6.3.2.1.7 OS_AUDIT_CLASS_KEY

The OS_AUDIT_CLASS_KEY audit message class is used when keys are generated, backed up, restored, opened, deleted, exported, imported, derived, wrapped and unwrapped and additional processes are performed. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
FC:0x068 SFC:0x05 key_init: <key group> [error code]	Initiating a key group Initiation: By the P11CAT or p11tool2 program when reinitializing a PKCS#11 slot, thereby deleting all key material.
FC:0x068 SFC:0x08 key_backup: <key info>, <key group>: <key name>: <key specifier> [error code]	Backing up a key Initiation: Call by the appropriate method of the keyCAT, P11CAT, p11tool2 or cxitool program although backing up a key is not a PKCS#11 function.
FC:0x068 SFC:0x09 key_restore: <key info>, <key group>: <key name>: <key specifier> [error code]	Restoring a key Initiation: Call by the appropriate method of the keyCAT, P11CAT, p11tool2 or cxitool program although restoring a key is not a PKCS#11 function.
FC:0x068 SFC:0x0B key_generate: <key info>, <key group>: <key name>: <key specifier> [error code]	Generating a key. Initiation: Call by the appropriate method of the csadm, P11CAT, p11tool2, cxitool or cngtool program.
FC:0x068 SFC:0x0C key_open: <key group>: <key name>: <key specifier> [error code]	Opening a key. Initiation: Call by the appropriate method of the csadm, CAT or P11CAT program.  Opening a key only creates audit log entries for SecurityServer/CryptoServer SDK versions earlier than 4.30 and CXI firmware module versions earlier than 2.4.0.0
FC:0x068 SFC:0x0D key_delete: <key info>, <key group>: <key name>: <key specifier> [error code]	Deleting a key. Initiation: Call by the appropriate method of the P11CAT, p11tool2, cxitool or cngtool program.
FC:0x068 SFC:0x0F key_prop_set: <key group>: <key name>: <key specifier> [error code]	Setting key properties. Initiation: Call by the appropriate method of the keyCAT, P11CAT program by changing key properties e.g., the label of a key.
FC:0x068 SFC:0x10 key_export: <key info>, <key group>: <key name>: <key specifier> [error code]	Exporting a key. Initiation: Call by the appropriate method of the keyCAT cngtool program.

Audit log entry	Description
FC:0x068 SFC:0x1 key_import: <key info>, <key group>: <key name>: <key specifier> [error code]	Importing a key. Initiation: Call by the appropriate method of the keyCAT cngtool program.
FC:0x068 SFC:0x19 obj_create: <key info>, <key group>: <key name>: <key specifier> [error code]	Creating an object. Initiation: Call by the appropriate method of the P11CAT or p11tool2 program.
FC:0x068 SFC:0x1A key_pair_gen: <key info>, <key group>: <public key name>: <public key specifier> / <private key name>: <private key specifier> [error code]	Generating a key pair. Initiation: Call by the appropriate method of the P11CAT or p11tool2.
FC:0x068 SFC:0x1B cxi_ext_obj_copy: <key info>, <key group>: <source key name>: <source key specifier> > <destination key group>: <destination key name>: <destination key specifier> [error code]	Copying an object. Initiation: Call by the appropriate method of the P11CAT or p11tool2 program.
FC:0x068 SFC:0x1C key_derive: <key info>, <key group>: <key name>: <key specifier> [error code]	Deriving a key. Initiation: Call by the appropriate method of the P11CAT or p11tool2 program.
FC:0x068 SFC:0x1D key_wrap: <key info>, <key group>: <key name>: <key specifier> [error code]	Wrapping a key. Initiation: Call by the appropriate method of the P11CAT or p11tool2 program.
FC:0x068 SFC:0x1E key_unwrap: <key info>, <key group>: <key name>: <key specifier> [error code]	Unwrapping a key. Initiation: Call by the appropriate method of the P11CAT or p11tool2 program.
FC:0x068 SFC:0x2B key_split: <key info>, <key group>: <key name>: <key specifier> [error code]	Splitting a key. Initiation: Appropriate call to the CXI firmware module.

Table 45: Audit log entries for OS_AUDIT_CLASS_KEY

6.3.2.1.8 OS_AUDIT_CLASS_AUTH_SUCCESS

The OS_AUDIT_CLASS_AUTH_SUCCESS audit message class is used when the authentication of a user succeeded. The following tables describe the corresponding audit log entries and the actions leading to them.

<i>Audit log entry</i>	<i>Description</i>
authentication (<authentication mechanism ID>) succeeded [error code]	Successful authentication. Initiation: csadm LogonPass or csadm LogonSign p11tool2 LoginUser cxitool LogonPass or cxitool LogonSign cngtool automatic login CAT: e.g. Login/Logout > Login Generic > User name > Login

Table 46: Audit log entries for OS_AUDIT_CLASS_AUTH_SUCCESS

6.3.2.1.9 OS_AUDIT_CLASS_AUTH_FAILED


The OS_AUDIT_CLASS_AUTH_FAILED audit message class is used when an authentication attempt fails. The following tables describe the corresponding audit log entries and the actions leading to them.





<i>Audit log entry</i>	<i>Description</i>
'<user>' authentication (<authentication mechanism ID>) failed, failure counter: <number> [error code]	Failed authentication without maximum value for failed authentication attempts set. Initiation: Call by the appropriate method of the csadm, CAT, P11CAT, p11tool2, cxitool or cngtool program.
'<user>' authentication (<authentication mechanism ID>) failed, <number> attempts left [error code]	Failed authentication with maximum value for failed authentication attempts set. Initiation: Call by the appropriate method of the csadm, CAT, P11CAT, p11tool2, cxitool or cngtool program.
'unknown' Authentication failed [error code]	Failed authentication due to a wrong user name. Initiation: Call by the appropriate method of the csadm, CAT, P11CAT, p11tool2, cxitool or cngtool program
'<user>' account locked, authentication refused [error code]	Too many failed authentication attempts. Initiation: Call by the appropriate method of the csadm, CAT, P11CAT, p11tool2, cxitool or cngtool program.


Table 47: Audit log entries for OS_AUDIT_CLASS_AUTH_FAILED

6.3.2.1.10 OS_AUDIT_CLASS_SYSTEM

The OS_AUDIT_CLASS_SYSTEM audit message class is used when the power-on self-test succeeded, the u.trust Anchor cHSM is shut down and some operations failed. The following tables describe the corresponding audit log entries and the actions leading to them.

<i>Audit log entry</i>	<i>Description</i>
FC: 0x088 SFC: Database '<database name>' corrupted (integrity failure detected)	Corrupted database
CMDS: Power-on Self-tests performed successfully	Successful power-on self-test Initiation: Windows command (shutdown -r -t 0) csadm Reset (not recommended), csadm Restart or csadm CSLReboot CAT: Manage > Reboot CryptoServer
[] FC:0x000 SFC: CryptoServer shut down	CryptoServer shutdown. cHSM shutdown Initiation: csadm Reset (not recommended), csadm ResetToBL , csadm Restart , csadm CSLShutdown or csadm CSLReboot) CAT: Manage > Reboot CryptoServer <div> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0.</div>
[] FC:0x000 SFC: DRBG: known answer test failed [<error code>]	The known answer test of the deterministic random bit generator (DRBG) has failed. This test verifies whether a device produces the expected output as a reaction to a given input. <div> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</div>

Audit log entry	Description
<p>[] FC:0x000 SFC: DRBG: initialization of '<profile name>' failed [<error code>]</p>	<p>Failed DRBG initialization The profile name is either 'Real' or 'Pseudo'.</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
<p>[] FC:0x000 SFC: DRBG: generation failed [<error code>]</p>	<p>Failed deterministic random number generation</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
<p>[] FC:0x000 SFC: DRBG: reseed failed [<error code>]</p>	<p>Failed reseed of the deterministic random bit generator</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
<p>[] FC:0x000 SFC: Module <function code> (<module name>) is corrupt [<error code>]</p>	<p>The firmware module has been detected as defective due to a failed integrity check. The function code is identical to the module ID.</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
<p>[] FC:0x000 SFC: TRNG: hardware failure [<error code>]</p>	<p>TRNG (true random number generator) hardware failure</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>

Audit log entry	Description
[] FC:0x000 SFC: TRNG: selftest failed [<error code>]	<p>Failed TRNG self-test</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
[] FC:0x000 SFC: TRNG: online test failed	<p>Poor statistical quality of a random number</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
FC:0x000 SFC: SHA-512 Known Answer Test failed	<p>Known Answer Test failed</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
[] FC:0x000 SFC: Unpack of module '<module name>' failed [<error code>]	<p>Unpacking a firmware module file, e.g., <code>adm.msc</code>, has failed</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>
[] FC:0x000 SFC: Loading of module '<module name>' failed [<error code>]	<p>Loading a firmware module file, e.g., <code>adm.msc</code>, has failed</p> <hr/> <p> This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> <hr/>


<i>Audit log entry</i>	<i>Description</i>
[] FC:0x000 SFC: Start function of module '<module name>' failed [error code]	<p>Start function of a firmware module file, e.g., <code>adm.msc</code>, has failed.</p> <div>  <p>This applies for V4.5.2.0 ≤ SMOS version V5.0.0.0 or SMOS version ≥ V5.5.4.0</p> </div>

Table 48: Audit log entries for OS_AUDIT_CLASS_SYSTEM

6.3.2.1.11 OS_AUDIT_CLASS_BACKUP_RESTORE

The OS_AUDIT_CLASS_BACKUP_RESTORE audit message class is used when a user is backed up or restored and a database entry is exported or imported. The following tables describe the corresponding audit log entries and the actions leading to them.

<i>Audit log entry</i>	<i>Description</i>
FC:0x083 SFC:0x0C Backup User '<user name>' [error code]	<p>Backing up a user. Initiation: <code>csadm BackupUser</code> or CAT: Manage > Manage User > Backup Users</p>
FC:0x083 SFC:0x0D Restore User '<user name>' (<Optional user name>, <authentication mechanism ID> <user's permission>) [error code]	<p>Restoring a user. Initiation: <code>csadm RestoreUser</code> or CAT Manage > Manage User > Restore Users This audit log entry is created if either OS_AUDIT_CLASS_BACKUP_RESTORE or OS_AUDIT_CLASS_USER is enabled.</p>
FC:0x087 SFC:0x18 Backup DB Entry (Database / Searchkey): '<DB name>' / '<user name>' [error code]	<p>Exporting a database entry. Initiation: <code>csadm BackupDatabase</code> or CAT Manage > Backup/Restore > Copy databases from Source CryptoServer to Backup directory > Select database > Add > Execute)</p>

Audit log entry	Description
FC:0x087 SFC:0x19 Restore DB Entry (Database / Searchkey): '<DB name>' / '<user name>' [error code]	Importing a database entry. Initiation: <code>csadm RestoreDatabase</code> or CAT Manage > Backup/Restore > Copy databases from Backup directory to Source CryptoServer > Select database > Add > Execute

Table 49: Audit log entries for OS_AUDIT_CLASS_BACKUP_RESTORE

6.3.2.1.12 OS_AUDIT_CLASS_CUSTOM_MASK

The OS_AUDIT_CLASS_CUSTOM_MASK audit message class is only used for extension implemented by the customer.

6.3.2.1.13 2020-0040a OS_AUDIT_CLASS_ALWAYS

If an audit message class name is set to OS_AUDIT_CLASS_ALWAYS, the audit log entry is written in any case.

The OS_AUDIT_CLASS_ALWAYS audit message class is used when the device is cleared, a certification mode (CC, FIPS) has to be handled, a new alarm is detected, the temperature exceeds its valid range etc. The following tables describe the corresponding audit log entries and the actions leading to them.

Audit log entry	Description
FC:0x087 SFC:0x15 Clear [error code]	Clearing the device Initiation: <code>csadm Clear</code> or CAT Manage > Clear
FC:0x087 SFC:0x1A change audit config param id: <ID> (<ID specifier>) from: <old> to: <new> [error code]	Changing audit log parameter. Initiation: <code>csadm SetAuditConfig</code> or CAT: Manage > Audit Log > Change configuration > OK
FIPS restrictions applied mode = <ON/OFF>, FIPS error state = <ON/OFF> [error code]	FIPS restrictions applied mode
FIPS mode = <ON/OFF>, FIPS error state = <ON/OFF> [error code]	FIPS mode. Initiation: Call by the appropriate method of the csadm or CAT.
[] FC:0x000 SFC: CryptoServer has entered FIPS mode	The CryptoServer has entered the FIPS mode. Initiation: Call by the appropriate method of the csadm or CAT.

Audit log entry	Description
[] FC:0x000 SFC: CryptoServer has left FIPS mode	The CryptoServer has left the FIPS mode. Initiation: Call by the appropriate method of the csadm or CAT program.
[] FC:0x000 SFC: CryptoServer ERASE to factory setting	CryptoServer ERASE to factory settings. Initiation: Call by the appropriate method of the csadm (including <code>clear=DEFAULT</code>) or CAT (including Manage > Clear to Factory Settings). Consider that all sensitive data is removed during this process. If you use the CryptoServer Simulator, use the <code>ALARM.curr</code> file for performing the External Erase, see "Sensor Detection" for details.
[] FC:0x000 SFC: New ALARM detected: <value> (<description>)	New alarm. Initiation: <ul style="list-style-type: none"> See, An Alarm and Its Consequences first, then Sensor Detection for a list of possible alarms. This chapter also describes how to create an alarm if you use a CryptoServer Simulator. In this case, you must perform a <code>csadm ... Restart</code> command or CAT > Manage > Reboot CryptoServer before the alarm is active. Consider that all sensitive data is removed during this process.
[] FC:0x000 SFC: Temperature exceeds valid range (<value>°C): Shutdown !	The temperature exceeds 62 °C or falls below -5°C, see 2.5.11, "Temperature Monitoring" for details.
[] FC:0x000 SFC: CryptoServer ERASE executed	ERASE executed. Initiation: See, An Alarm and Its Consequences first, then follow the instructions in <i>Performing an External Erase</i> in the <i>CryptoServer - CAT Manual</i> . Consider that all sensitive data is removed during this process.

Table 50: Audit log entries for OS_AUDIT_CLASS_ALWAYS

6.3.2.1.14 OS_AUDIT_CLASS_ACTION_NEEDED

The OS_AUDIT_CLASS_ACTION_NEEDED audit message class is used to alert the user, that host must be updated. The following tables describe the corresponding audit log entries and the actions leading to them.


<i>Audit log entry</i>	<i>Description</i>
user <username> authenticated with old HMAC mech and host must be updated	<p>A user that has HMAC authentication and logs in using old host software (without PBKDF) and new firmware (with PBKDF) is successfully logged in and a new entry is added to the audit log.</p> <div> We encourage the user to upgrade the host software and use a more secure mechanism of HMAC authentication.</div>

Table 51: Audit log entries for OS_AUDIT_CLASS_ACTION_NEEDED

6.3.2.2 Signed Audit Logs

The u.trust Anchor cHSM can generate signed audit log files on the computer on which csadm is running.

The audit log signature key, see [Audit Log Signature Key](#), is used for this process. The generated files have extra long audit log file names (see *Audit Log File Names* in the [CryptoServer - csadm Manual](#)). For details about how to generate and use this key, see *Generating and Verifying Signed Audit Log Files* in the [u.trust Anchor - csadm Manual](#).

7 Troubleshooting

7.1 Alarm Treatment

An alarm can not be triggered on the cHSM itself, but will always occur on the u.trust Anchor device itself. If a cHSM is accessible, this means no alarm state is present on the u.trust Anchor device.

8 Contact Address for Support Queries

If an error occurs while operating the u.trust Anchor cHSM, read [Troubleshooting](#).

If the error still occurs, prepare diagnostic information in a `.txt` file with the information retrieved via `gladm system-get-info`.

If you have any further questions on u.trust Anchor, 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
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.

9 References

<i>Reference</i>	<i>Title/Company</i>	<i>Document Number</i>
	u.trust Anchor - Administrator Guide / Utimaco IS GmbH.	2020-0035
	u.trust Anchor - Containerized Hardware Security Module (cHSM) - User Manual/ Utimaco IS GmbH.	2020-0034
	u.trust Anchor - csadm Manual / Utimaco IS GmbH.	2021-0037
	CryptoServer - CryptoServer CSP and CNG Key Storage Provider/Utimaco IS GmbH	2008-0002
	CryptoServer – PKCS#11 P11CAT Manual / Utimaco IS GmbH.	M013-0001-en
	CryptoServer - PKCS#11 p11tool2 - Reference Manual / Utimaco IS GmbH.	2012-0004
	CryptoServer PKCS#11 R3 - Developer Guide	2012-0007
[SMOS]	CryptoServer – Operating System SMOS – SMOS Version ≥ 2.5.0.0 – Interface Specification / Utimaco IS GmbH	2008-0001
[ANSSI]	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. Available: https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000024668816	
[ANSI-X9.62]	ANSI X9.62-2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA) / ANSI (American National Standards Institute)	
[BP]	RFC 5639, ECC Brainpool Standard Curves and Curve Generation, March 2010	
[FIPS140-2]	FIPS PUB 140-2, Security Requirements for Cryptographic Modules / National Institute of Standards and Technology (NIST), May 2001	
[FIPS186-4]	FIPS PUB 186-4, Digital Signature Standard / National Institute of Standards and Technology (NIST), July 2013	
[NISTSP800]	NIST Special Publication 800-56A Revision 3, Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography / National Institute of Standards and Technology (NIST), April 2018	
[RFC7748]	RFC 7748: Elliptic Curves for Security / Internet Research Task Force (IRTF), January 2016, ISSN 2070-1721, including Errata ID 4730 reported and verified on 2016-07-05	

Reference	Title/Company	Document Number
[SEC2]	SEC2: Recommended Elliptic Curve Domain Parameters – Certicom Research – January 27, 2010, Version 2.0	
[SP80056A]	SP 800-56A Rev. 3: Recommendation for Pair-Wise Key- Establishment Schemes Using Discrete Logarithm Cryptography/National Institute of Standards and Technology (NIST), April 2018	