AUSTRIACARD

ACDIS Sapphire PKCS#11 Developer Manual

Version: 1.2.0.0		AUSTRIACARD
Changed on:	02/10/2024 11:41 AM	Created on: 05/09/2023
Status:	FINAL	

Table of Content

1. GENERAL INFORMATION

1.1. COPYRIGHT

Austria Card Plastikkarten und Ausweissysteme GmbH is the sole owner of the information, knowledge and representations contained in this document. The documentation and the information, knowledge and representations contained within may not be provided to third parties, neither complete nor in part, directly nor indirectly, published nor otherwise dispersed. The assertion of all related rights, especially in the case of distribution of patents, is strictly reserved to Austria Card. The transfer of the documentation is no entitlement for a license or use.

1.2. DOCUMENT HISTORY

Version	Date	Author	Description
1.0.0.0	05/09/2023	AUSTRIACARD, Markus Punz	First Draft v1.0.0.0
1.0.0.1	22/09/2023	AUSTRIACARD, Markus Punz	Changed “ACOSID” to “ACDIS”
1.0.0.2	23/10/2023	AUSTRIACARD, Markus Punz	Clarification CKA_ID in 7.2.1, 7.2.2.1, 7.2.2.2 Clarification CKA_LABEL in 7.2.1, 7.2.2.1 Added C_SetAttributeValue in 7.2.7 Clarification SO-PIN in 7.2.4 Clarification CKA_MODIFIABLE in 7.2.2.2

			Added C_CreateObject for CKO_DATA in 7.2.2.4
1.0.0.3	06/11/2023	AUSTRIACARD, Markus Punz	Added support for SSCD applications
1.0.0.4	22/11/2023	AUSTRIACARD, Markus Punz	Changed term “PKCS#11 application” to “PKCS#11 generic application” New Token-Suffixes for SSCD applications
1.0.0.5	23/11/2023	AUSTRIACARD, Markus Punz	No Changes in Doc. New Version due to PKCS#11-Library Version 1.3
1.0.0.6	18/12/2023	AUSTRIACARD, Markus Punz	Hyphenation removed for shell commands. New Version due to PKCS#11-Library Version 1.4
1.0.0.7	17/01/2024	AUSTRIACARD, Markus Punz	Added installation for Apple-Silicon (ARM) - 3.3.1 and 3.3.2
1.0.0.8	19/01/2024	AUSTRIACARD, Markus Punz	No Changes in Doc. New Version due to PKCS#11-Library Version 1.5
1.1.0.0	03/04/2024	AUSTRIACARD, Markus Punz	Changes due to PKCS#11-Library Version 1.6 Added configuration for using ACDIS smart cards with Acrobat Reader (- see 10.2) Added support for CA-Certificates

1.2.0.0	02/10/2024	AUSTRIACARD, Markus Punz	Changes due to PKCS#11-Library Version 1.7 Added 7.2.8 Added 7.2.9

1.3. PKCS#11-LIBRARY HISTORY

Version	Date	Author	Description
1.0	23/10/2023	AUSTRIACARD	First release.
1.1	06/11/2023	AUSTRIACARD	SSCD Upgrade
1.2	22/11/2023	AUSTRIACARD	SSCD Fix – added suffix for Token-Labels of SSCD applications, e.g.: - ACDIS.SSCD1.PIN1-active or - ACDIS.SSCD1.PIN1-inactive or - ACDIS.SSCD1.PIN1-locked
1.3	23/11/2023	AUSTRIACARD	Internal Changes of ATR-Check.
1.4	18/12/2023	AUSTRIACARD	Logging fixes. Error handling of Mobility card reader.
1.5	19/01/2024	AUSTRIACARD	Fix: Import certificates > 1024 bytes for SSCD-keys
1.6	03/04/2024	AUSTRIACARD	Primary Token of Generic PKCS#11 application returns a public vendor defined object with the minidriver-configuration (Generic-Mode vs. SSCD-Mode)

			Added support for CA-Certificates for Generic-PKCS11 application and SSCD-applications
1.7	02/10/2024	AUSTRIACARD	Each Key Length supports hash-length that are longer than the size of the key. This means: ECC-Key P256 supports signing of hash values calculated by SHA-384 and SHA-512 ECC-Key P384 supports signing of hash values calculated by SHA-512 This is supported by truncating the hash values to the key length. C_GetMechanismList for SSCD-Tokens: With SSCD there is exactly one SSCD key per virtual slot. This can be an RSA key or an ECC key However, the PKCS#11 module only knows whether it is RSA or ECC after a PIN verification (C_Login). This means that when logged out, all possible algorithms are returned - i.e. RSA and ECC

			In difference to the logged-in state, where all ECC algorithms are returned if it is an ECC key and all RSA algorithms if it is an RSA key Fixed CKR_DEVICE_ERROR when, for example an ECC-Algorithm is requested for an RSA-key or an RSA-Algorithm is requested von an ECC-Key. Now CKR_MECHANISM_INVALID is returned in this case. C_SignInit for SSCD-Tokens: The module returns CKR_USER_NOT_LOGGED_IN in case the PIN is not authenticated, and returns CKR_KEY_HANDLE_INVALID in case the PIN is authenticated but caller passed an invalid key handle.

1.4. DISCLAIMER OF LIABILITY

Austria Card guarantees for a period of twenty-four months from the time of delivery that the Software essentially corresponds to the program description in the accompanying written material with regard to its functionality.

Austria Card points out that the Software is qualified as a one-off service. Necessary updates can be obtained via the same channels through which the Software was obtained as long as the warranty applies.

Austria Card points out that according to the state of the art it is not possible to produce computer Software completely error-free.

If a defect occurs, the defect and its appearance must be described in such detail in a written notice of defect that a review of the defect (e.g. submission of error messages) is feasible and the exclusion of an operating error (e.g. specification of the work steps) is possible.

If the notice of defects proves to be justified, the licensee sets AUSTRIA CARD a reasonable deadline for subsequent performance. The licensee informs AUSTRIA CARD which type of supplementary performance - improvement of the delivered or delivery of a new, defect-free item - he wishes. However, AUSTRIA CARD is entitled to refuse the selected supplementary performance if this can only be carried out with disproportionate costs for it and if the other type of supplementary performance would not entail any significant disadvantages for the licensee. AUSTRIA CARD may also refuse subsequent performance altogether if it can only be carried out at disproportionate cost to it.

In order to carry out the supplementary performance, AUSTRIA CARD is entitled to two attempts for the same or directly related defect within the period set by the licensee. After the second failed attempt at subsequent performance, the licensee may withdraw from the contract or reduce the license fee. The right of withdrawal or reduction can already be exercised after the first unsuccessful attempt at subsequent performance, if a second attempt within the set period is not reasonable for the licensee. If subsequent performance has been refused under the conditions set out

above, the licensee is entitled to the right of reduction or withdrawal immediately.

If the licensee has made a claim against AUSTRIA CARD under warranty, and it turns out that either there is no defect or the asserted defect does not oblige AUSTRIA CARD to provide a warranty, the licensee must reimburse all expenses incurred by AUSTRIA CARD if he is grossly negligent or intentional responsible for the use of AUSTRIA CARD

A guarantee that the Software is suitable for the purposes of the licensee and cooperates with the licensee's existing Software is excluded.

Beyond this warranty, AUSTRIA CARD shall only be liable for a period of two years from delivery of the Software in the event of intent and gross negligence in accordance with the statutory provisions. In the event of slight negligence, AUSTRIA CARD shall only be liable if an essential contractual obligation is violated or if there is a case of delay or impossibility. In the event of liability arising from slight negligence, this liability shall be limited to such damages that are foreseeable or typical. Liability for the lack of the guaranteed quality, due to fraudulent intent, for personal injury and the General Data Protection Agreement remains unaffected.

In the event of a claim against AUSTRIA CARD under warranty or liability, contributory negligence on the part of the user must be taken into account appropriately, in particular in the case of insufficient error messages or insufficient data backup. Insufficient data backup exists in particular if the licensee has failed to take precautions against external influences, in particular against computer viruses and other phenomena that can endanger individual data or an entire database, by means of appropriate, state-of-the-art security measures.

Under no circumstances shall AUSTRIA CARD or its affiliates, partners, suppliers or licensors be liable for any indirect, incidental, consequential, special or exemplary damages arising out of or in connection with your access or use of or inability to access or use the Software and any third party content and services, whether or not the damages were foreseeable and whether or not company was advised of the possibility of such damages.

2. OVERVIEW

This document is the developer manual for the AUSTRIACARD ACDIS Sapphire PKCS#11 library.

The ACDIS Sapphire PKCS#11 library implements common cryptographic interfaces.

The interfaces implemented are

- PKCS #11 Version 2.20

It supports the following crypto tokens:

- AUSTRIACARD ACDIS Smartcard

Access to the smartcard is implemented via smart card terminals over the PC/SC interface.

Note:

- PIN entry on the integrated keypad is not supported for card readers with a pin pad. The PIN is entered exclusively via the PC keyboard.

2.1. SUPPORTED OPERATING SYSTEMS

- Microsoft Windows 10 22H2 – 32 Bit
- Microsoft Windows 10 22H2 – 64 Bit
- Microsoft Windows 10 LTSC 2021 – 32 Bit
- Microsoft Windows 10 LTSC 2021 – 64 Bit
- Microsoft Windows 11 22H2 – 64 Bit
- Microsoft Windows 11 23H2 – 64 Bit
- Ubuntu Linux 20.04 LTS
- Ubuntu Linux 22.04 LTS
- Debian 11 / 12
- macOS Monterey (Version 12)
- macOS Ventura (Version 13)
- macOS Sonoma (Version 14)

3. INSTALLATION

Installation depends on the operating system:

3.1. WINDOWS 32-BIT / 64-BIT

The PKCS#11 library is supplied with its own installation program (.exe).

Note:

- The Visual C++ Redistributable for Visual Studio 2015-2022 is required. If not available, it will be installed during the installation of the PKCS#11 module.

3.2. LINUX

The installation is carried out using the “Homebrew” package manager.

To do this, Homebrew must be installed for the first time if it is not already installed

3.2.1. Prerequisites for Homebrew

Homebrew requires the tools „curl“ and „git“.

If not available, these must be installed first.

If applicable execute:

```
sudo apt install curl
sudo apt install git
```

3.2.2. Install Homebrew

Please execute:

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

After succcessful install execute:

- test -d ~/.linuxbrew && eval "$(~/.linuxbrew/bin/brew shellenv)"

- test -d /home/linuxbrew/.linuxbrew && eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
- test -r ~/.bash_profile && echo "eval \"\$($(brew --prefix)/bin/brew shellenv)\"" >> ~/.bash_profile
- echo "eval \"\$($(brew --prefix)/bin/brew shellenv)\"" >> ~/.profile

3.2.3. Install PKCS#11 Library via Homebrew

Please execute:

- brew tap austriacard/acdislinux
- brew install acdislinux

Note:

The above commands install the PKCS#11 library (= libacdis-pkcs11.so) in the directory /home/linuxbrew/.linuxbrew/Cellar/acdislinux/<<Version>>/

During the initial installation, a number of dependencies are installed. Please note that the installation process may take some time.

A symbolic link to the libacdis-pkcs11.so file is created in the directory /home/linuxbrew/.linuxbrew/lib .

In order for programs to be able to load the .so library, the above lib directory must be included in the LD_LIBRARY_PATH, e.g.:

- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/linuxbrew/.linuxbrew/lib

We recommend configuring this path in the user profile so that it is set automatically every time you log in. To do this, please do the following:

- echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/linuxbrew/.linuxbrew/lib" >> ~/.profile

The PKCS#11 library uses PC/SC to access the card reader. The PC/SC smart card daemon must therefore be installed (if not available):

- Install PC/SC smart card daemon:

  sudo apt-get install pcscd

- Start of PC/SC smart card daemon:

  sudo service pcscd start

3.2.4. Uninstall PKCS#11 Library

- brew uninstall acdislinux
- brew untap austriacard/acdislinux

3.3. MACOS

The installation is also carried out under macOS using the “Homebrew” package manager. To do this, Homebrew must first be installed if it is not already installed.

3.3.1. Install Homebrew

For Intel-Mac and Apple-Silicon please execute:

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Please also do the following on Apple-Silicon (ARM) – must be omitted on Intel-Mac:

- (echo; echo 'eval "$(/opt/homebrew/bin/brew shellenv)"') >> ~/.zprofile
- eval "$(/opt/homebrew/bin/brew shellenv)"

3.3.2. Install PKCS#11 Library via Homebrew

Please execute:

- brew tap austriacard/acdismac
- brew install acdismac

Note:

On Intel-Mac the above commands install the PKCS#11 library (= acdis-pkcs11.dylib) in the directory /usr/local/Cellar/acdismac/<<Version>>/

A symbolic link to the acdis-pkcs11.dylib file is created in the /usr/local/lib directory.

On Apple-Silicon the PKCS#11 library is installed in the directory /opt/homebrew/Cellar/acdismac/<<Version>>/

A symbolic link to the acdis-pkcs11.dylib file is created in the /opt/homebrew/lib directory.

In order for programs to be able to load the .dylib library, the above lib directory must be included in the DYLD_LIBRARY_PATH

3.3.3. Uninstall PKCS#11 Library

- brew uninstall acdismac
- brew untap austriacard/acdismac

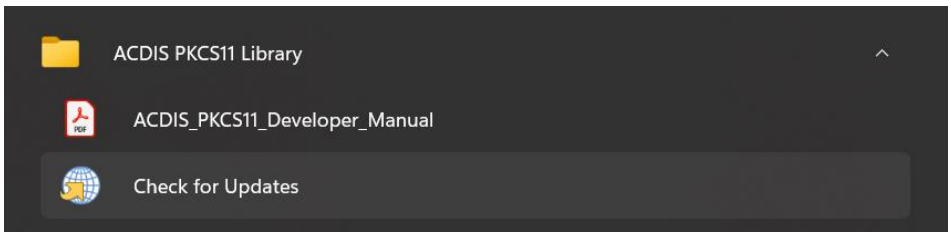
3.4. ALL OPERATING SYSTEMS

Please install the card reader driver recommended by the manufacturer for your device.

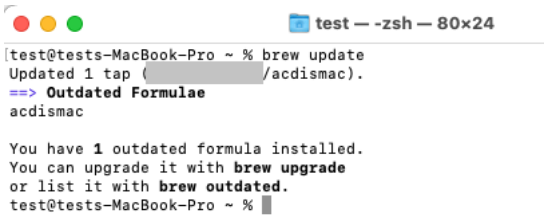
4. LIBRARY UPDATES

4.1. MANUALLY TRIGGERED UPDATES

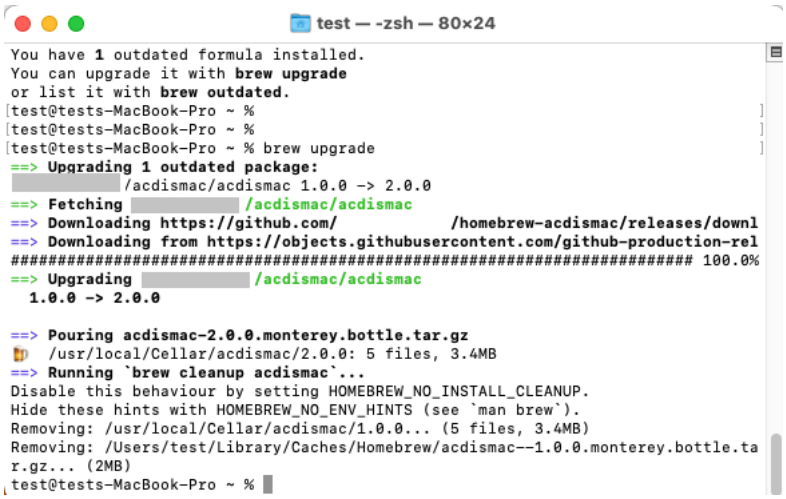
To check whether there is a new version, Windows users can run the “Check for updates” function in the Start menu:



Linux/Mac users can run the “brew update” function on the command line. This checks whether there is a new version:



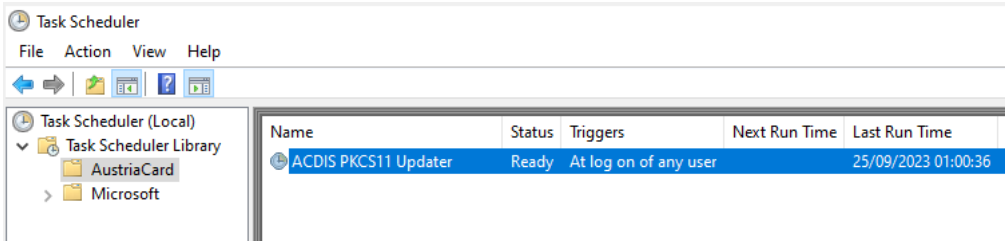
If a new version is found, it can be installed with "brew upgrade":



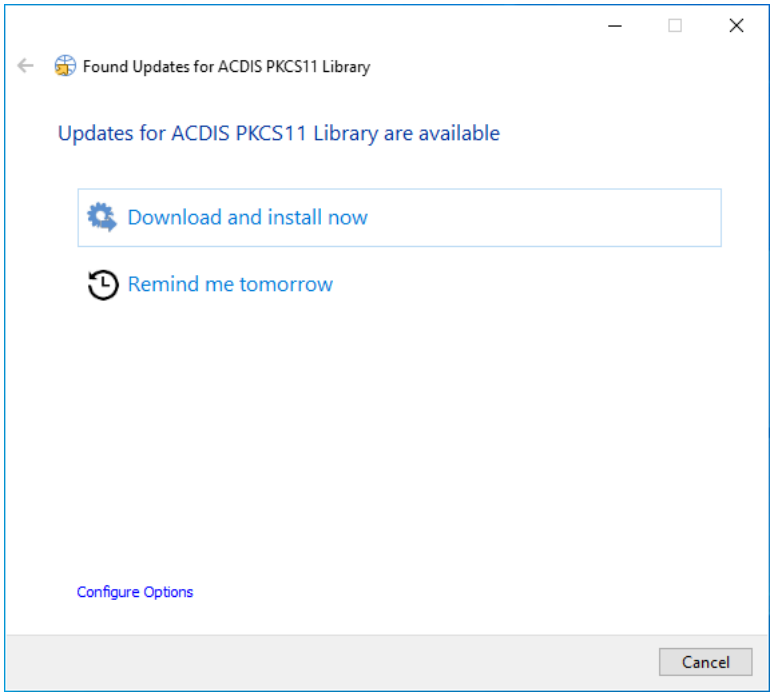
4.2. CONFIGURE AUTOMATIC UPDATES

4.2.1. Windows

The ACDIS PKCS#11 Library installer configures a task in the Windows scheduler that automatically checks after each login whether there is a new version of the library.



If so, the user can decide whether he wants to install the update or not.



4.2.2. MacOS

The Homebrew package manager can be configured to install upgrades automatically. To do this, please run the following in a terminal window:

```
brew tap homebrew/autoupdate
mkdir -p $HOME/Library/LaunchAgents
```

If you want to check for updates every 24h (= default)

```
brew autoupdate start --upgrade
```

Alternatively: If you want to check for updates every hour (= 3.600 seconds)

Please specify the desired update-interval in seconds

```
brew autoupdate start 3600 --upgrade
```

To stop automatic updates you can do the following:

```
brew autoupdate stop
```

4.2.3. Linux

Automatic updates can be configured using shell commands that are executed automatically after login.

To do this, create a linuxbrewUpdate.sh in the user HOME with the following content:

```
#!/bin/bash
brew update
brew upgrade
```

Then run the following command in the terminal window so that the update script is started automatically after every login:

```
echo "$HOME/linuxbrewUpdate.sh &" >> ~/.profile
```

Note:

Other variants, for example in which the above script is periodically executed as a cron job, are also possible.

We won't go into this in more detail here.

5. PKCS#11 LIBRARY

Under Windows the ACDIS Sapphire PKCS#11 library is implemented as a WIN32/64 Dynamic Link Library (DLL).

The 32-bit DLL is named acdis-pkcs11.dll. On 32-bit Windows-systems it is installed in the Windows\System32 folder. On 64-bit Windows-systems it is installed in the Windows\SysWOW64 folder.

The 64-bit DLL is named acdis-pkcs11-64.dll and is installed in the Windows\System32 folder (- only on Windows 64-bit systems).

Under Linux the ACDIS Sapphire PKCS#11 library is implemented as a 64-bit shared-object library. A symbolic Link to the so-library is named libacdis-pkcs11.so and is installed in /home/linuxbrew/.linuxbrew/lib .

Under macOS the ACDIS Sapphire PKCS#11 library is implemented as a 64-bit dynamic library. A symbolic Link to the dylib-library is named acdis-pkcs11.dylib and is installed in /usr/local/lib for Intel-Mac and /opt/homebrew/lib for Apple-Silicon.

5.1. PKCS#11 – LIBRARY INFO

The structure CK_INFO contains common information about the PKCS#11 library.

Name	Content	Description
cryptokiVersion	2.20	Version of PKCS#11 specification
manufacturerID	AustriaCard (www.austriacard.at)	Library manufacturer
flags	0	RFU
libraryDescription	ACDIS PKCS#11-Module	Description of library
libraryVersion	X.X	Version of library

6. PKCS#11 APPLICATIONS

The ACDIS PKCS#11 library supports the following types of applications:

- PKCS#11 generic applications
- SSCD applications

An ACDIS smart card can contain 0 or a maximum of 1 PKCS#11 generic applications and 0 or a maximum of 4 SSCD applications.

6.1. PKCS#11 GENERIC APPLICATION

The PKCS#11 generic application can be personalized yourself. When delivered, 5 USER-PINs are created as well as an additional SO-PIN.

The PINs are initialized with default values:

USER-PIN	Default Value
ROLE_USER (= Primary PIN of smart card)	1234
PIN#3 (Auth)	3333
PIN#4 (DigSig)	4444
PIN#5 (Enc)	5555
PIN#6 (NonRep)	6666

SO-PIN (hex-encoded):
\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30
(= 16 ASCII zeros “0000000000000000”)

Note:

The SO-PIN is 16-Byte long and is internally a 3DES-Key. 3-DES has a parity bit on the least significant digit which is not used for cryptography.

SO-PINs that only differ in this parity bit are the same.

This means the SO-PIN (hex-encoded):
\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30

is the same as
\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31
because it only differs in the last bit.

6.1.1. Slot Info

The structure CK_SLOT_INFO contains information about the smart card reader assigned to the respective slot.

The ACDIS Sapphire PKCS#11 library implements one slot for each attached smart card reader.

Name	Content	Description
slotDescription	e.g. “OMNIKEY CardMan 3x21 0”	Name of smart card reader
manufacturerID	AustriaCard	Fixed
flags	CKF_REMOVABLE_DEVICE (0x02) CKF_HW_SLOT (0x04) CKF_TOKEN_PRESENT (0x01)	token may be removed from terminal token is implemented in hardware set if token inserted
hardwareVersion	0.0	Fixed
firmwareVersion	0.0	Fixed

Because PKCS#11 is not intended to be used with more than one user PIN, a compliant solution for multi-PIN scenarios was introduced called virtual slots (see PKCS#11 v2.10).
This enables a smart card to provide multiple user PINs for key material. To applications it appears that there are multiple slots and tokens despite being a single physical smart card.

Virtual slots/tokens in detail:

As mentioned before a PKCS#11 generic application provides 5 USER-PINs:

- ROLE_USER (= Primary PIN of smart card)
- PIN#3 (Auth)
- PIN#4 (DigSig)
- PIN#5 (Enc)
- PIN#6 (NonRep)

1-10 asymmetric key pairs can be generated on ACDIS smart cards. It can be assigned which of the above PINs is used to protect a private key.
Each private key has exactly one of these PINs associated with it. However, multiple keys can be protected with the same PIN.

Example:

ECC Key#0 256 bits is protected with PIN “ROLE_USER”
RSA Key#1 2048 is protected with “PIN#3 (Auth)”

In this case, the PKCS#11 library generates 2 slots/tokens for the card reader in which the ACDIS smart card is located. Let's assume that this is the card reader "OMNIKEY CardMan 3x21".
The following slots are then offered via PKCS#11 (= field “slotDescription” in CK_SLOT_INFO):

- OMNIKEY CardMan 3x21 0 (= Primary-Slot)
- OMNIKEY CardMan 3x21 0.2 (= Virtual-Slot)

The CK_TOKEN_INFO can be read using C_GetTokenInfo.
The labels are set as follows:

CK_SLOT_INFO slotDescription	CK_TOKEN_INFO label	Description
OMNIKEY Card-Man 3x21 0	ACDIS	Primary-Token All keys that are protected with the PIN "ROLE_USER" can be used via this slot (according to our example the ECC Key#0 256 bits)
OMNIKEY Card-Man 3x21 0.2	ACDIS.Auth	Virtual-Token All keys that are protected with the PIN "PIN#3 (Auth)" can be used via this slot (according to our example the RSA Key#1 2048 bits)

Since up to 5 PINs can be assigned, a maximum of 5 slots are generated per card reader:

- OMNIKEY CardMan 3x21 0 (= Primary-Slot)
- OMNIKEY CardMan 3x21 0.2 (= Virtual-Slot#1)
- OMNIKEY CardMan 3x21 0.3 (= Virtual-Slot#2)
- OMNIKEY CardMan 3x21 0.4 (= Virtual-Slot#3)
- OMNIKEY CardMan 3x21 0.5 (= Virtual-Slot#4)

Depending on the assigned PIN, the tokens have the following labels:

CK_TOKEN_INFO label	Description
ACDIS	Primary-Token. Provides all keys protected with PIN "ROLE_USER"
ACDIS.Auth	Virtual-Token. Provides all keys protected with PIN#3 (Auth)
ACDIS.DigSig	Virtual-Token. Provides all keys protected with PIN#3 (Dig-Sig)

ACDIS.Enc	Virtual-Token. Provides all keys protected with PIN#3 (Enc)
ACDIS.NonRep	Virtual-Token. Provides all keys protected with PIN#3 (Non-Rep)

By default, the PKCS#11 library always generates the primary Token "ACDIS". Additional virtual tokens are only generated if they contain keys.
This means that if an ACDIS smart card only contains keys that are protected with the PIN "ROLE_USER" then only 1 slot/token (- the primary-slot/token) is created. No virtual slots/tokens are generated for the remaining PINs (PIN#3-PIN#6) that have no key assigned. This means that as long as a PIN has not been assigned a key, no C_Login can be made with this PIN.

This behavior can be overridden with the environment variable

- PKCS11_SHOW_ALL_VIRTUAL_SLOTS

In this case, the PKCS#11 Library offers one virtual slot/token per PIN - even if no key is protected with this PIN.

Note:

It is only relevant that the environment variable exists. The actual value does not matter and is not checked.

6.1.2. Token Info

The structure CK_TOKEN_INFO contains information about either the (virtual) smart card inserted in the respective smart card reader.

Name	Content	Description
label	One of the following: • ACDIS • ACDIS.Auth • ACDIS.DigSig	token name

	• ACDIS.Enc • ACDIS.NonRep	
manufacturerID	AustriaCard	token manufacturer
model	ACDIS	token model
serialNumber	\xNN\xNN…. \xNN	16 Bytes serial number
flags	CKF_LOGIN_REQUIRED CKF_RNG CKF_TOKEN_INITIALIZED CKF_USER_PIN_INITIAL-IZED	• login is required • token has random number generator • token is initialized • PIN is initialized
ulMaxSessionCount	CK_EFFECTIVELY_INFINITE	max. number of PKCS#11 sessions
ulSessionCount	CK_UNAVAILABLE_ INFORMATION	number of current PKCS#11 sessions
ulMaxRWSessionCount	CK_EFFECTIVELY_INFINITE	max. number of PKCS#11 RW-sessions
ulRWSessionCount	CK_UNAVAILABLE_ INFORMATION	number of current PKCS#11 RW-sessions
ulMaxPINLength	14	max. PIN length in bytes
ulMinPINLength	4	min. PIN length in bytes
ulTotalPublicMemory	CK_UNAVAILABLE_ INFORMATION	total amount of memory for public PKCS#11 objects

ulFreePublicMemory	CK_UNAVAILABLE_ INFORMATION	free memory for public PKCS#11 objects
ulTotalPrivateMemory	CK_UNAVAILABLE_ INFORMATION	total amount of memory for private PKCS#11 objects
ulFreePrivateMemory	CK_UNAVAILABLE_ INFORMATION	free memory for pri-vate PKCS#11 objects
hardwareVersion	X.X	token hardware ver-sion
firmwareVersion	X.X	token firmware version
utcTime	Not used	Current time

6.2. SSCD APPLICATION

SSCD applications are pre-personalized. Each application can contain 1-5 QES keys.

Note:

If more than 5 QES keys are required, several SSCD applications can be pre-personalized.

Every SSD application has qualified signature keys that are already generated during smart card pre-personalization. This means that when the smart card is delivered, all QES keys are already present on the card. Each signature key has its own signature PIN. These PINs are inactive when delivered and must be activated for the first time.

Each signature key and the associated signature PIN form a virtual PKCS#11 token.

Virtual slots/tokens in detail:

As mentioned before a SSCD Application has a maximum of 5 signature PINs/Keys:

- PIN#1 – for signature key #1
- PIN#2 – for signature key #2
- PIN#3 – for signature key #3
- PIN#4 – for signature key #4
- PIN#5 – for signature key #5

Let's assume a concrete SSCD application contains 3 signature PINs / Keys:
In this case, the PKCS#11 library generates 3 slots/tokens for the card reader in which the ACDIS smart card is located. Let's assume that this is the card reader "OMNIKEY CardMan 3x21".

The following slots are then offered via PKCS#11 (= field “slotDescription” in CK_SLOT_INFO):

- OMNIKEY CardMan 3x21 0 (= Primary-Slot)
- OMNIKEY CardMan 3x21 0.2 (= Virtual-Slot)
- OMNIKEY CardMan 3x21 0.3 (= Virtual-Slot)

The CK_TOKEN_INFO can be read using C_GetTokenInfo.
The labels are set as follows:

CK_SLOT_INFO slotDescription	CK_TOKEN_INFO label	Description
OMNIKEY Card-Man 3x21 0	ACDIS.SSCD1.PIN1	Slot/Token for signature PIN#1 / Key#1
OMNIKEY Card-Man 3x21 0.2	ACDIS.SSCD1.PIN2	Slot/Token for signature PIN#2 / Key#2
OMNIKEY Card-Man 3x21 0.3	ACDIS.SSCD1.PIN3	Slot/Token for signature PIN#3 / Key#3

Depending on the assigned PIN, the tokens have the following labels:

CK_TOKEN_INFO label	Description
ACDIS.SSCD<<n>>.PIN1	Token for signature PIN#1 / Key#1
ACDIS.SSCD<<n>>.PIN2	Token for signature PIN#2 / Key#2
ACDIS.SSCD<<n>>.PIN3	Token for signature PIN#3 / Key#3
ACDIS.SSCD<<n>>.PIN4	Token for signature PIN#4 / Key#4
ACDIS.SSCD<<n>>.PIN5	Token for signature PIN#5 / Key#5

<<n>> … Identifier of SSCD application:
1 -> for SSCD application number 1
2 -> for SSCD application number 2
etc.

As long as the signature PIN is inactive, the virtual slot is also inactive.
This is shown as a suffix in the token label:

- ACDIS.SSCD<<n>>.PIN<<m>>-inactive

If the PIN is activated (- details see 8.1 Activation of SSCD Signature PIN) the following token-label is shown:

- ACDIS.SSCD<<n>>.PIN<<m>>-**active**

And if the PIN is blocked:

- ACDIS.SSCD<<n>>.PIN<<m>>-**blocked**

6.2.1. Token Info

The structure CK_TOKEN_INFO contains information about either the (virtual) smart card inserted in the respective smart card reader.

Name	Content	Description
label	One of the following: ACDIS.SSCD<n>.PIN<m>-<suffix>	token name <n> … Identifier of SSCD application (1-n) <m> … Identifier of PIN (1-m) <suffix>: - active - inactive - blocked
manufacturerID	AustriaCard	token manufacturer
model	ACDIS	token model
serialNumber	"XXXXXXXXXXXXXXXX"	16 ASCII-Digits (= card identification data)
flags	CKF_LOGIN_REQUIRED CKF_RNG CKF_TOKEN_INITIALIZED CKF_WRITE_PROTECTED	• login is required • token has random number generator • token is initialized

	If signature PIN is already activated: CKF_USER_PIN_INITIAL-IZED	• token is write pro-tected • PIN is initialized
ulMaxSessionCount	CK_EFFECTIVELY_INFINITE	max. number of PKCS#11 sessions
ulSessionCount	CK_UNAVAILABLE_ INFORMATION	number of current PKCS#11 sessions
ulMaxRWSessionCount	CK_EFFECTIVELY_INFINITE	max. number of PKCS#11 RW-sessions
ulRWSessionCount	CK_UNAVAILABLE_ INFORMATION	number of current PKCS#11 RW-sessions
ulMaxPINLength	12	max. PIN length in bytes
ulMinPINLength	6	min. PIN length in bytes
ulTotalPublicMemory	CK_UNAVAILABLE_ INFORMATION	total amount of memory for public PKCS#11 objects
ulFreePublicMemory	CK_UNAVAILABLE_ INFORMATION	free memory for public PKCS#11 objects
ulTotalPrivateMemory	CK_UNAVAILABLE_ INFORMATION	total amount of memory for private PKCS#11 objects

ulFreePrivateMemory	CK_UNAVAILABLE_ INFORMATION	free memory for private PKCS#11 objects
hardwareVersion	X.X	token hardware version
firmwareVersion	X.X	token firmware version
utcTime	Not used	Current time

In order to create qualified signatures, the following steps must be carried out:

1)

Activate signature PINs – for details see 8.1 Activation of SSCD Signature PIN

2)

Read public signature keys – for details see 8.2 Reading of SSCD Public-Key

3)

Request a qualified certificate for the public signature key and write certificate. Empty certificates are already created on the smart card. This means the attribute CKA_VALUE is NULL. Certificate data can be written with C_CreateObject.

– for details see 8.3 WRITE SSCD signature Certificates

Finally, qualified signatures can be created – details see 8.4 Creation of qualified signatures.

6.3. PKCS#11 MECHANISMS

6.3.1. RSA using PKCS#1

The CKM_RSA_PKCS mechanism is used for signing of data or encryption and decryption of keys.

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	2048	Minimum key size
ulMaxKeySize	4096	Maximum key size
flags	CKF_HW CKF_SIGN CKF_VERIFY CKF_ENCRYPT CKF_DECRYPT	Implemented in hardware Mechanism is used for signing and encryption / decryption of keys

Note:

CKM_RSA_PKCS is not supported for RSA-keys of SSCD applications.

6.3.2. RSA using OEAP

The CKM_RSA_OAEP mechanism is used for encryption and decryption of keys.

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	2048	Minimum key size
ulMaxKeySize	4096	Maximum key size
flags	CKF_HW CKF_ENCRYPT CKF_DECRYPT	Implemented in hardware Mechanism is used for encryption / decryption of keys

CK_RSA_PKCS_OAEP_PARAMS

Name	Description
hashAlg	Hash-algorithm – CKM_SHA1 or CKM_SHA256 or CKM_SHA384 or CKM_SHA512 Default CKM_SHA256 if not set
pSourceData	ignored - card uses empty string as label
ulSourceDataLen	ignored - card uses empty string as label
source	ignored - must be "0" because label not supported
mgf	ignored - card uses same hash-function for MGF as specified in hashAlg-parameter

Note:

CKM_RSA_OAEP is not supported for RSA-keys of SSCD applications.

6.3.3. RSA using PSS

The CKM_RSA_PSS mechanism is used for signing of data.

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	2048	Minimum key size
ulMaxKeySize	4096	Maximum key size
flags	CKF_HW CKF_SIGN CKF_VERIFY	Implemented in hardware Mechanism is used for signing.

Note:

CKM_RSA_PSS is not supported for RSA-keys of SSCD applications.

6.3.4. RSA with Hashing using PKCS#1

The following mechanisms are used for hashing and signing of data (= signing on card, digest calculation in software):

- CKM_SHA1_RSA_PKCS
- CKM_SHA256_RSA_PKCS
- CKM_SHA384_RSA_PKCS
- CKM_SHA512_RSA_PKCS

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	2048	Minimum key size
ulMaxKeySize	4096	Maximum key size
flags	CKF_HW	Implemented in hardware

	CKF_SIGN CKF_VERIFY	Mechanism is used for signing.

Note:

The above mechanisms are not supported for RSA-keys of SSCD applications.

6.3.5. RSA with Hashing using PSS

The following mechanisms are used for hashing and signing of data (= signing on card, digest calculation in software):

- CKM_SHA1_RSA_PSS (- not supported for RSA-keys of SSCD applications)
- CKM_SHA256_RSA_PSS
- CKM_SHA384_RSA_PSS
- CKM_SHA512_RSA_PSS

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	2048	Minimum key size
ulMaxKeySize	4096	Maximum key size
flags	CKF_HW CKF_SIGN CKF_VERIFY	Implemented in hardware Mechanism is used for signing.

6.3.6. ECDSA

The CKM_ECDSA mechanism is used for signing of data.

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	256	Minimum key size
ulMaxKeySize	521	Maximum key size
flags	CKF_HW CKF_SIGN CKF_VERIFY	Implemented in hardware Mechanism is used for signing.

If the given data exceeds the ECC key size the PKCS#11 library truncates it at the key-length.

Note:

CKM_ECDSA is not supported for ECC-keys of SSCD applications.

6.3.7. ECDSA with Hashing

The following mechanisms are used for hashing and signing of data (= signing on card, digest calculation in software):

- CKM_ECDSA_SHA1 (- not supported for ECC-keys of SSCD applications)
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384 (only-supported by ECC key-length >= 384 bits)
- CKM_ECDSA_SHA512 (only-supported by ECC key-length = 521 bits)

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	256	Minimum key size

ulMaxKeySize	521	Maximum key size
flags	CKF_HW CKF_SIGN CKF_VERIFY	Implemented in hardware Mechanism is used for signing.

6.3.8. Hashing

The following mechanisms are used for hashing of data (- digest calculation is done in software):

- CKM_SHA1
- CKM_SHA256
- CKM_SHA384
- CKM_SHA512

CK_MECHANISM_INFO

Name	Content	Description
ulMinKeySize	CKM_UNAVAILABLE_INFORMATION	
ulMaxKeySize	CKM_UNAVAILABLE_INFORMATION	
flags	CKF_DIGEST	Mechanism is used for hashing.

6.4. PKCS#11 OBJECTS

6.4.1. Certificates

Name	Content	Description
CKA_CLASS	CKO_CERTIFICATE	PKCS#11 type
CKA_CERTIFI-CATE_TYPE	CKC_X_509	Type of certificate
CKA_TRUSTED	FALSE	Trusted-state of certificate
CKA_CERTIFI-CATE_CATEGORY	1 (=Token user)	Category of certificate
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	FALSE	Object is public accessible
CKA_LABEL	For PKCS#11 ge-neric application: mscp\ksc0**N** or mscp\kxc0**N** For SSCD applica-tion: C.CH.SIG**N**	Name of object N… 0-9
CKA_ID	1-N	PKCS#11 ID of certificate. This ID can be used to tie together public and private keys and certificates.
CKA_VALUE		ASN.1 DER encoded certificate NULL if certificate is empty.
CKA_MODIFIABLE	FALSE	Object may not be changed

CKA_ISSUER	Issuer	ASN.1 DER encoded content of certificate issuer Not available if certificate is empty.
CKA_SERIAL_NUM-BER	Serial number	ASN.1 DER encoded content of certificate serial number Not available if certificate is empty.
CKA_SUBJECT	subject	ASN.1 DER encoded content of certificate subject Not available if certificate is empty.

6.4.2. CA-Certificates

Name	Content	Description
CKA_CLASS	CKO_CERTIFICATE	PKCS#11 type
CKA_CERTIFI-CATE_TYPE	CKC_X_509	Type of certificate
CKA_TRUSTED	TRUE	Trusted-state of certificate
CKA_CERTIFI-CATE_CATEGORY	2 (= Authority)	Category of certificate
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	FALSE	Object is public accessible
CKA_LABEL	C.CA.CERTN	Name of object N… 0-n
CKA_ID	1-N	PKCS#11 ID of CA certificate.
CKA_VALUE		ASN.1 DER encoded certificate

		NULL if certificate is empty.
CKA_MODIFIABLE	FALSE	Object may not be changed
CKA_ISSUER	Issuer	ASN.1 DER encoded content of certificate issuer Not available if certificate is empty.
CKA_SERIAL_NUM-BER	Serial number	ASN.1 DER encoded content of certificate serial number Not available if certificate is empty.
CKA_SUBJECT	subject	ASN.1 DER encoded content of certificate subject Not available if certificate is empty.

6.4.3. RSA Public-Keys

Name	Content	Description
CKA_CLASS	CKO_PUBLIC_KEY	PKCS#11 type
CKA_KEY_TYPE	CKK_RSA	Type of key
CKA_TRUSTED	FALSE	Trusted-state of key
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	For PKCS#11 generic application: FALSE For SSCD application:	Object is public accessible Object is not public accessible C_Login is required

	TRUE	
CKA_LABEL	For PKCS#11 generic applications: XXXX For SSCD-Applications: K.CH.SIGN	Given name of object during C_GenerateKeypair or C_CreateObject 16 Bytes N… Logical number of key.
CKA_ID	1-N	PKCS#11 ID of key. This ID can be used to tie together public and private keys and certificates.
CKA_MODIFIABLE	FALSE	Object may not be changed
CKA_ENCRYPT	TRUE	Key supports encryption
CKA_WRAP	FALSE	Key does not support wrapping
CKA_VERIFY	TRUE	Key supports verification
CKA_VERIFY_RECOVER	FALSE	Key does not support verification with message recovery
CKA_DERIVE	FALSE	Key does not support key derivation
CKA_MODULUS	XX..XX	Modulus of key
CKA_MODULUS_BITS	2048 / 3072 / 4096	Key length of key
CKA_PUBLIC_EXPONENT	X'010001'	Public exponent of key

CKA_LOCAL	TRUE	Means: Key was generated on the token. Note: This value is always set to TRUE.
CKA_KEY_GEN_MECH-ANISM	CKM_UNAVAILA-BLE_ INFORMATION	mechanism used to generate key

6.4.4. ECC Public-Keys

Name	Content	Description
CKA_CLASS	CKO_PUBLIC_KEY	PKCS#11 type
CKA_KEY_TYPE	CKK_EC	Type of key
CKA_TRUSTED	FALSE	Trusted-state of key
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	For PKCS#11 generic appli-cation: FALSE For SSCD application: TRUE	Object is public accessible Object is not public acces-sible C_Login is required
CKA_LABEL	For PKCS#11 generic appli-cations: XXXX	Given name of object dur-ing C_GenerateKeypair or C_CreateObject 16 Bytes N… Logical number of key.

	For SSCD-Applications: K.CH.SIGN	
CKA_ID	1-N	PKCS#11 ID of key. This ID can be used to tie together public and private keys and certificates.
CKA_MODIFIABLE	FALSE	Object may not be changed
CKA_ENCRYPT	FALSE	Key supports encryption
CKA_WRAP	FALSE	Key does not support wrapping
CKA_VERIFY	TRUE	Key supports verification
CKA_VERIFY_RECOVER	FALSE	Key does not support verification with message recovery
CKA_DERIVE	FALSE	Key does not support key derivation
CKA_EC_PARAMS	• 06082A8648CE3D030107 • 06052B81040022 • 06052B81040023	ASN.1 DER encoded ObjectID of EC-curve: • secp256r1 • secp384r1 • secp521r1
CKA_EC_POINT	XX..XX	ASN.1 DER encoded EC Public-Key: 04 << len >> 04 x || y

CKA_LOCAL	TRUE	Means: Key was generated on the token. Note: This value is always set to TRUE.
CKA_KEY_GEN_MECH-ANISM	CKM_UNAVAILABLE_ INFORMATION	mechanism used to generate key

6.4.5. RSA Private-Keys

Name	Content	Description
CKA_CLASS	CKO_PRI-VATE_KEY	PKCS#11 type
CKA_KEY_TYPE	CKK_RSA	Type of key
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	TRUE	Object is not public accessible
CKA_LABEL	For PKCS#11 ge-neric applications: XXXX For SSCD-Applica-tions: K.CH.SIGN	Given name of object during C_GenerateKeypair or C_Cre-ateObject 16 Bytes N… Logical number of key.
CKA_ID	1-N	PKCS#11 ID of key. This ID can be used to tie together public and private keys and certificates.

CKA_MODIFIABLE	FALSE	Object may not be changed
CKA_SENSITIVE	TRUE	Key is secret
CKA_DECRYPT	For PKCS#11 generic applications: TRUE / FALSE For SSCD-Applications: FALSE	Key supports decryption Depends on how the key was created.
CKA_UNWRAP	FALSE	Key does not support unwrapping
CKA_SIGN	TRUE	Key supports signature
CKA_SIGN_RECOVER	FALSE	Key does not support signature with message recovery
CKA_DERIVE	FALSE	Key does not support key derivation
CKA_MODULUS	XX..XX	Modulus of key
CKA_PUBLIC_EXPONENT	X'010001'	Public exponent of key
CKA_LOCAL	TRUE	Means: Key was generated on the token. Note: This value is always set to TRUE.
CKA_EXTRACTABLE	FALSE	Key may not be exported from token
CKA_NEVER_EXTRACTABLE	TRUE	Key may never be exported from token

CKA_ALWAYS_SENSITIVE	TRUE	Key is always secret
CKA_KEY_GEN_MECHANISM	CKM_UNAVAILABLE_INFORMATION	mechanism used to generate key
CKA_ALWAYS_AUTHENTICATE	For PKCS#11 generic applications: FALSE For SSCD-Applications: TRUE	Key does not require authentication for each use. Key requires authentication for each use.

6.4.6. ECC Private-Keys

Name	Content	Description
CKA_CLASS	CKO_PRIVATE_KEY	PKCS#11 type
CKA_KEY_TYPE	CKK_EC	Type of key
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	TRUE	Object is not public accessible
CKA_LABEL	For PKCS#11 generic applications: XXXX	Given name of object during C_GenerateKeypair or C_CreateObject 16 Bytes N… Logical number of key.

	For SSCD-Applications: K.CH.SIGN	
CKA_ID	1-N	PKCS#11 ID of key. This ID can be used to tie together public and private keys and certificates.
CKA_MODIFIABLE	FALSE	Object may not be changed
CKA_SENSITIVE	TRUE	Key is secret
CKA_DECRYPT	FALSE	Key does not support decryption
CKA_UNWRAP	FALSE	Key does not support unwrapping
CKA_SIGN	TRUE	Key supports signature
CKA_SIGN_RECOVER	FALSE	Key does not support signature with message recovery
CKA_DERIVE	FALSE	Key does not support key derivation
CKA_LOCAL	TRUE	Means: Key was generated on the token. Note: This value is always set to TRUE.
CKA_EXTRACTABLE	FALSE	Key may not be exported from token
CKA_NEVER_EXTRACTABLE	TRUE	Key may never be exported from token

CKA_ALWAYS_SENSITIVE	TRUE	Key is always secret
CKA_KEY_GEN_MECHANISM	CKM_UNAVAILABLE_INFORMATION	mechanism used to generate key
CKA_ALWAYS_AUTHENTICATE	For PKCS#11 generic applications: FALSE For SSCD-Applications: TRUE	Key does not require authentication for each use. Key requires authentication for each use.
CKA_EC_PARAMS	• 06082A8648CE3D030107 • 06052B81040022 • 06052B81040023	ASN.1 DER encoded ObjectID of EC-curve: • secp256r1 • secp384r1 • secp521r1

6.4.7. Vendor-defined Configuration

The Primary-Token of the generic PKCS#11 application may return a public object with a vendor-defined object.

Name	Content	Description
CKA_CLASS	CKO_VENDOR_DEFINED	PKCS#11 type
CKA_TOKEN	TRUE	Token object
CKA_PRIVATE	FALSE	Object is public accessible
CKA_MODIFIABLE	TRUE	The CKA_VALUE attribute may be changed.
CKA_APPLICATION	“Config”	Fixed.
CKA_VALUE		Configuration-Settings for Minidriver-Mode (- can be set via PKCS#11-Manager) NULL if empty

7. PKCS#11 FUNCTIONS

The following table is based on PKCS#11 specification version 2.20. It lists all functions as defined by the standard.

7.1.1. General purpose functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_Initialize	Yes	Yes
C_Finalize	Yes	Yes
C_GetInfo	Yes	Yes
C_GetFunctionList	Yes	Yes

7.1.2. Slot and token management functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_GetSlotList	Yes	Yes
C_GetSlotInfo	Yes	Yes
C_GetTokenInfo	Yes	Yes
C_WaitForSlotEvent	Yes	Yes
C_GetMechanismList (- see details 7.2.9)	Yes	Yes
C_GetMechanismInfo	Yes	Yes
C_InitToken	Yes	No
C_InitPIN	Yes	Yes

		see details 7.2.5 C_InitPIN
C_SetPIN	Yes	Yes

7.1.3. Session management functions

Function	Supported for PKCS#11 generic application	Supported for SSCD applica-tion
C_OpenSession	Yes	Yes
C_CloseSession	Yes	Yes
C_CloseAllSessions	Yes	Yes
C_GetSessionInfo	Yes	Yes
C_GetOperationState	No	Yes
C_SetOperationState	No	Yes
C_Login For details on authentication state see 7.2.9	Yes	Yes
C_Logout For details on authentication state see 7.2.9	Yes	Yes

7.1.4. Object management functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_CreateObject	Yes	Yes (- only for CKO_CERTIFICATE)
C_CopyObject	No	No
C_DestroyObject	Yes	No
C_GetObjectSize	Yes	Yes
C_GetAttributeValue	Yes	Yes
C_SetAttributeValue (- see details 7.2.7 C_SetAttributeValue)	No	No
C_FindObjectsInit	Yes	Yes
C_FindObjects	Yes	Yes
C_FindObjectsFinal	Yes	Yes

7.1.5. Encryption and Decryption functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_EncryptInit	Yes	Yes
C_Encrypt	Yes	Yes
C_EncryptUpdate	No	No
C_EncryptFinal	No	No

C_DecryptInit	Yes	No
C_Decrypt	Yes	No
C_DecryptUpdate	No	No
C_DecryptFinal	No	No

7.1.6. Message digesting functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_DigestInit	Yes	Yes
C_Digest	Yes	Yes
C_DigestUpdate	Yes	Yes
C_DigestKey	No	No
C_DigestFinal	Yes	Yes

7.1.7. Signing and MACing functions / functions for verifying signatures and MACs

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_SignInit	Yes	Yes
C_Sign (- see details 7.2.8)	Yes	Yes
C_SignUpdate	Yes	Yes
C_SignFinal	Yes	Yes
C_SignRecoverInit	No	No

C_SignRecover	No	No
C_VerifyInit	Yes	Yes
C_Verify	Yes	Yes
C_VerifyUpdate	Yes	Yes
C_VerifyFinal	Yes	Yes
C_VerifyRecoverInit	No	No
C_VerifyRecover	No	No

7.1.8. Dual-purpose cryptographic functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_DigestEncryptUpdate	No	No
C_DecryptDigestUpdate	No	No
C_SignEncryptUpdate	No	No
C_DecryptVerifyUpdate	No	No

7.1.9. Key management functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_GenerateKey	No	No
C_GenerateKeyPair (- see details 7.2.1 C_GenerateKeyPair)	Yes	No
C_WrapKey	No	No
C_UnwrapKey	No	No

C_DeriveKey	No	No

7.1.10. Random number generation functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_SeedRandom	No	No
C_GenerateRandom	Yes	Yes

7.1.11. Parallel function management functions

Function	Supported for PKCS#11 generic application	Supported for SSCD application
C_GetFunctionStatus Returns: CKR_FUNCTION_NOT_PARALLEL	Yes	Yes
C_CancelFunction Returns: CKR_FUNCTION_NOT_PARALLEL	Yes	Yes

7.2. DETAILS ON PKCS#11 FUNCTIONS

7.2.1. C_GenerateKeyPair

Not supported for SSCD applications – only for PKCS#11 generic application.

C_GenerateKeyPair always generates a public key at the same time as the private key.

It supports the following mechanisms (CK_MECHANISM):

- CKM_EC_KEY_PAIR_GEN
- CKM_RSA_PKCS_KEY_PAIR_GEN

Requirements:
The prerequisite for C_GenerateKeyPair is that you have successfully performed a C_Login on the primary token (- with the PIN "ROLE_USER").
Only then key pairs can be created.

The public key template supports the following attributes (CK_ATTRIBUTE):

Note:

- The general attributes apply to both RSA keys and ECC keys. Accordingly, the RSA attributes only apply to RSA keys and the ECC attributes to ECC keys.

- Applies to attributes of type CK_Bool:

  False = 0

  True = 1

  All other values are not permitted for CK_Bool.

Attribute	Content
General-Attributes:	
CKA_TOKEN	Optional. If provided must be True
CKA_VERIFY	Optional. If provided must be True
CKA_KEY_TYPE	Optional. Either CKK_RSA or CKK_EC. Must match CK_MECHANISM if provided.
CKA_CLASS	Optional. If provided must be CKO_PUBLIC_KEY
CKA_ENCRYPT	Optional. True or False (Default: False) Ignored for EC-Keys (- EC-Keys do not support Encryption)
CKA_WRAP	Optional. If provided must be False
RSA-Attributes:	
CKA_MODULUS_BITS	Must. Supported values: 2048 / 3072 / 4096
CKA_PUBLIC_EXPO-NENT	Optional. If provided must match 65537
ECC-Attributes:	
CKA_EC_PARAMS	Must. Supported values: • \x06\x08\x2A\x86\x48\xCE\x3D\x03\x01\x07 (secp256r1 aka prime256v1) • \x06\x05\x2B\x81\x04\x00\x22 (secp384r1) • \x06\x05\x2B\x81\x04\x00\x23 (secp521r1)

The private key template supports the following attributes (CK_ATTRIBUTE):

Attribute	Content
General-Attributes:	
CKA_TOKEN	Optional. If provided must be True
CKA_SIGN	Optional. If provided must be True
CKA_KEY_TYPE	Optional. Either CKK_RSA or CKK_EC. Must match CK_MECHANISM if provided.
CKA_CLASS	Optional. If provided must be CKO_PRIVATE_KEY
CKA_ID	Optional. If provided: The PKCS#11 module checks whether the specified CKA_ID is in the valid range 1-10. If this CKA_ID is still free, i.e. if there is no existing private key with this CKA_ID, the CKA_ID provided by the caller is used for the new key pair. In all other cases a new CKA_ID is generated - e.g.: • If the caller did not specify a CKA_ID • If the provided CKA_ID is not in the valid range 1-10 • If the provided CKA_ID is already used by another key pair
CKA_LABEL	Optional. If provided: • The provided CKA_LABEL must be unique. If not, a unique value is generated by the PKCS#11 module. • Length must not exceed 39 bytes If not provided: • PKCS#11 library generates a default value
CKA_DECRYPT	Optional. True or False (Default: False) Ignored for EC-Keys (- EC-Keys do not support Decryption)

CKA_SENSITIVE	Optional. If provided must be True
CKA_EXTRACTABLE	Optional. If provided must be False
CKA_UNWRAP	Optional. If provided must be False
CKA_DERIVE	Optional. If provided must be False
CKA_PRIVATE	Optional. If provided must be True

Additional Checks:

- If CKA_KEY_TYPE is provided for public and private key template they must have the same value (both CKK_RSA or CKK_EC)
- CKA_ENCRYPT of the public key template must match CKA_DECRYPT of the private key template (both True or False)

Note:

- Further attributes can be specified within the templates but they will be ignored.

A maximum of 10 key pairs can be generated.

7.2.1.1 C_GenerateKeyPair for virtual tokens

Also in this case a successful C_Login on the primary token (- with the PIN "ROLE_USER") is required first.
After that the virtual slot/token for which the key pair is to be generated must be selected and a C_Login with the respective PIN (= PIN#3-PIN#6) of the token must be performed. Then the key pair can be created.

If the virtual slot/token is not offered by the PKCS#11 library (because the token is still empty and no key pair has been created for it), then the environment variable PKCS11_SHOW_ALL_VIRTUAL_SLOTS must be set. Then all virtual slots/tokens are visible - even if they do not yet contain any keys.
See chapter 6.1.1 Slot Info for details.

Note:

- C_GenerateKeyPair always needs successful authentication (C_Login) with PIN “ROLE_USER”. This is due to compatibility reasons to the Microsoft Windows Minidriver architecture.

7.2.1.2 Source-Code example

```
CK_SLOT_ID          slotid_primary_token, slotid_virtual_token_1;
CK_OBJECT_HANDLE    hndPrivateKey, hndPublicKey;
CK_MECHANISM        mech_genrsa = { CKM_RSA_PKCS_KEY_PAIR_GEN, 0, 0 };
CK_ATTRIBUTE        publicKeyTemplate[20];
int                 publicKeyAttributes = 0;

CK_ATTRIBUTE        privateKeyTemplate[20];
int                 privateKeyAttributes = 0;

rc = p11->C_GetSlotList(TRUE, NULL, &slots);
if (rc != CKR_OK) exit(1);

slotlist = (CK_SLOT_ID_PTR)malloc(sizeof(CK_SLOT_ID) * slots);
rc = p11->C_GetSlotList(TRUE, slotlist, &slots);
if (rc != CKR_OK) exit(1);

slotid_primary_token = *(slotlist);
slotid_virtual_token_1 = *(slotlist + 1);

// Open session to primary token
rc = p11->C_OpenSession(slotid_primary_token, CKF_RW_SESSION | CKF_SERIAL_SESSION,
NULL, NULL, &session);
if (rc != CKR_OK) exit(1);

// Login to primary token
rc = p11->C_Login(session, CKU_USER, (CK_UTF8CHAR*)"1234", 4);
if (rc != CKR_OK) exit(1);

p11->C_CloseSession(session);

// Open session to virtual token
rc = p11->C_OpenSession(slotid_virtual_token_1, CKF_RW_SESSION |
CKF_SERIAL_SESSION, NULL, NULL, &session);
if (rc != CKR_OK) exit(1);

// Login to virtual token
rc = p11->C_Login(session, CKU_USER, (CK_UTF8CHAR*)"3333", 4);
if (rc != CKR_OK) exit(1);

// Create RSA-2048 on virtual token
CK_ULONG keysize = 2048;
publicKeyAttributes = 0;
privateKeyAttributes = 0;
publicKeyTemplate[publicKeyAttributes].type = CKA_MODULUS_BITS;
publicKeyTemplate[publicKeyAttributes].pValue = &keysize;
publicKeyTemplate[publicKeyAttributes].ulValueLen = sizeof(keysize);
publicKeyAttributes++;

rc = p11->C_GenerateKeyPair(session, &mech_genrsa,
       publicKeyTemplate, publicKeyAttributes,
       privateKeyTemplate, privateKeyAttributes,
       &hndPublicKey, &hndPrivateKey);

if (rc != CKR_OK) exit(1);

free(slotlist);
```

7.2.2. C_CreateObject

Requirements for PKCS#11 generic applications:
The prerequisite for C_CreateObject is that you have successfully performed a C_Login on the primary token (- with the PIN "ROLE_USER").
Only then objects can be created.

In the following, 2 variants of C_CreateObject are distinguished:

- C_CreateObject of type CKO_PRIVATE_KEY
- C_CreateObject of type CKO_CERTIFICATE

7.2.2.1 C_CreateObject – CKO_PRIVATE_KEY

Note:

- C_CreateObject – CKO_PRIVATE_KEY is only supported for PKCS#11 generic applications. Not for SSCD-Applications.

- C_CreateObject of type CKO_PUBLIC_KEY is generally not supported. A public key cannot be created independently of a private key.
  Only private keys can be created. The associated public key is generated automatically.

The template supports the following attributes (CK_ATTRIBUTE):

Attribute	Content
General-Attributes:	
CKA_CLASS	Must. Value must be CKO_PRIVATE_KEY
CKA_TOKEN	Optional. If provided must be True
CKA_SIGN	Optional. If provided must be True

CKA_ID	Optional. If provided: The PKCS#11 module checks whether the specified CKA_ID is in the valid range 1-10. If this CKA_ID is still free, i.e. if there is no existing private key with this CKA_ID, the CKA_ID provided by the caller is used for the new key pair. In all other cases a new CKA_ID is generated - e.g.: • If the caller did not specify a CKA_ID • If the provided CKA_ID is not in the valid range 1-10 • If the provided CKA_ID is already used by another key pair
CKA_LABEL	Optional. If provided: • The provided CKA_LABEL must be unique. If not, a unique value is generated by the PKCS#11 module. • Length must not exceed 39 bytes If not provided: • PKCS#11 library generates a default value
CKA_DECRYPT	Optional. True or False (Default: False) Ignored for EC-Keys (- EC-Keys do not support Encryption)
CKA_SENSITIVE	Optional. If provided must be True
CKA_EXTRACTABLE	Optional. If provided must be False
CKA_UNWRAP	Optional. If provided must be False
CKA_DERIVE	Optional. If provided must be False
CKA_PRIVATE	Optional. If provided must be True
RSA-Attributes:	
CKA_MODULUS	Must. Public-Modulus must be provided.

CKA_PUBLIC_EXPO-NENT	Optional. If provided must match 65537
CKA_PRIME_1	Must. Value of p
CKA_PRIME_2	Must. Value of q
CKA_PRIVATE_EXPO-NENT	Optional. Ignored if provided.
CKA_EXPONENT_1	Must. Value of dp
CKA_EXPONENT_2	Must. Value of dq
CKA_COEFFICIENT	Must. Value of qinv
CKA_KEY_TYPE	Optional. If provided must be CKK_RSA
ECC-Attributes:	
CKA_EC_PARAMS	Must. Supported values: • \x06\x08\x2A\x86\x48\xCE\x3D\x03\x01\x07 (secp256r1 aka prime256v1) • \x06\x05\x2B\x81\x04\x00\x22 (secp384r1) • \x06\x05\x2B\x81\x04\x00\x23 (secp521r1)
CKA_EC_POINT	Optional. Ignored if provided. The PublicKey is calculated by the PKCS#11 library via point-multiplication Q=d*G
CKA_VALUE	Must. Value of Private-Key
CKA_KEY_TYPE	Optional. If provided must be CKK_EC

Note:

PKCS#11 states that imported keys must have the following attribute:

- CKA_LOCAL: False
- CKA_ALWAYS_SENSITIVE: False
- CKA_NEVER_EXTRACTABLE: False

The PKCS#11 library cannot differentiate if a key is generated on card or imported. In contrast to the PKCS#11-specification in both cases (C_GenerateKeypair and C_CreateObject) the mentioned attributes have the following values:

- CKA_LOCAL: True
- CKA_ALWAYS_SENSITIVE: True
- CKA_NEVER_EXTRACTABLE: True

For details see 6.4.5 and 6.4.6.

A maximum of 10 private keys can be imported.

7.2.2.2 C_CreateObject – CKO_CERTIFICATE

C_CreateObject of type CKO_CERTIFICATE is supported for both PKCS#11 generic applications and SSCD applications.

Requirements for PKCS#11 generic applications:
The prerequisite for C_CreateObject is that you have successfully performed a C_Login on the primary token (- with the PIN "ROLE_USER").
Only then objects can be created.

Requirements for SSCD applications:
The prerequisite for C_CreateObject is that you have successfully performed a C_Login with the assigned signature PIN.

Note:

User certificates (CKA_CERTIFICATE_CATEGORY=1) and CA certificates (CKA_CERTIFICATE_CATEGORY=2) can be created. User certificates are certificates whose private/public key is also stored on the card.
CA certificates are typically the parent certificates of the user certificates. The private key of a CA certificate is not stored on the card.

For user-certificates the template supports the following attributes (CK_ATTRIBUTE):

Attribute	Content
CKA_CLASS	Must. Value must be CKO_CERTIFICATE
CKA_VALUE	Must. DER-encoded certificate.
CKA_ID	Optional. For PKCS#11 generic application: If the caller provides a CKA_ID, the module searches for a key pair with that CKA_ID. If one is present, the module checks whether the Modulus/EC_Point of the provided certificate matches the Modulus/EC_Point of the found key pair. If yes: Private key and certificate match. In the next step, the module checks whether a certificate already exists for this CKA_ID - if so, an error is returned that the certificate already exists. If there is no certificate, it will be imported into the card with the provided CKA_ID. If no CKA_ID is provided by the caller, or if a CKA_ID is provided but a suitable Private Key for that CKA_ID cannot be found in the above step, a search for a suitable Private-Key is performed. The Private Key must have the same modulus/EC_Point as the certificate to be imported.If a matching private key is found, the certificate is imported with the CKA_ID of the Private Key.

	If no Private Key was found for the certificate's Modulus/EC_Point, an error is returned. For SSCD application: Ignored. The existing keypair must have same modulus/exponent (- for RSA) or EC_Point/EC_Params (- for EC) as the imported certificate.
CKA_TOKEN	Optional. If provided must be True
CKA_PRIVATE	Optional. If provided must be False
CKA_MODIFIABLE	Ignored. Object cannot be changed after creation
CKA_TRUSTED	Optional. If provided must be False
CKA_CERTIFICATE_TYPE	Optional. If provided must be CKC_X_509
CKA_CERTIFICATE_CATEGORY	Optional. If provided must be 1 (= token user) If not provided value 1 is assumed by default.
CKA_LABEL	Ignored. Because CKA_LABEL is automatically generated. For PKCS#11 generic application: mscp\ksc0N or mscp\kxc0N For SSCD application: C.CH.SIGN
CKA_SUBJECT	Ignored if provided because CKA_SUBJECT is taken from certificate.
CKA_ISSUER	Ignored if provided because CKA_ISSUER is taken from certificate.

CKA_SERIAL_NUMBER	Ignored if provided because CKA_SERIAL_NUMBER is taken from certificate.

For CA-certificates the template supports the following attributes (CK_ATTRIBUTE):

Attribute	Content
CKA_CLASS	Must. Value must be CKO_CERTIFICATE
CKA_VALUE	Must. DER-encoded certificate.
CKA_ID	Ignored.
CKA_TOKEN	Optional. If provided must be True
CKA_PRIVATE	Optional. If provided must be False
CKA_MODIFIABLE	Ignored. Object cannot be changed after creation
CKA_TRUSTED	Optional. If provided must be True
CKA_CERTIFI-CATE_TYPE	Optional. If provided must be CKC_X_509
CKA_CERTIFI-CATE_CATEGORY	Must. Must be 2 (= Authority). If not provided value 1 (= Token user) is assumed by default.
CKA_LABEL	Ignored. Because CKA_LABEL is automatically generated. C.CA.CERTn
CKA_SUBJECT	Ignored if provided because CKA_SUBJECT is taken from certificate.
CKA_ISSUER	Ignored if provided because CKA_ISSUER is taken from certificate.
CKA_SERIAL_NUMBER	Ignored if provided because CKA_SERIAL_NUMBER is taken from certificate.

7.2.2.3 Special case: C_CreateObject for virtual tokens of PKCS#11 generic applications

Also in this case a successful C_Login on the primary token (- with the PIN "ROLE_USER") is required first.
After that the virtual slot/token for which the object is to be imported must be selected and a C_Login with the respective PIN (= PIN#3-PIN#6) of the token must be performed. Then the object can be imported.
The procedure is exactly the same as shown in section 7.2.1.2 Source-Code example.

If the virtual slot/token is not offered by the PKCS#11 library (because the token is still empty and no key pair has been created / imported for it), then the environment variable PKCS11_SHOW_ALL_VIRTUAL_SLOTS must be set. Then all virtual slots/tokens are visible - even if they do not yet contain any keys.
See chapter 6.1.1 Slot Info for details.

7.2.2.4 C_CreateObject – CKO_DATA

Objects of type CKO_DATA are currently always created as session objects. They are not written to the token.

7.2.3. C_DestroyObject

For PKCS#11 generic application the following PKCS#11 objects can be deleted:

- CKO_PRIVATE_KEY
- CKO_PUBLIC_KEY
- CKO_CERTIFICATE (Token-user and Authority)

For SSCD application the following PKCS#11 objects can be deleted:

- CKO_CERTIFICATE (Authority)

Requirements for PKCS#11 generic applications:
The prerequisite for C_DestroyObject is that you have successfully performed a C_Login on the primary token (- with the PIN "ROLE_USER").
Only then objects can be deleted.

Requirements for SSCD applications:
The prerequisite for C_ DestroyObject is that you have successfully performed a C_Login with the assigned signature PIN.

Note:

A C_DestroyObject on the hndPrivateKey leads to the private key, public key and associated certificate being deleted physically (ie the certificate is automatically deleted as well, if present).
Separate deletion of public key and certificate is not necessary.

A C_DestroyObject of public keys is supported. However, it is only deleted from the PKCS#11 session objects. Not on physical card itself.
This means that the public key is there again when the PKCS#11 library is re-initialized because it was not physically deleted.

For C_DestroyObject on virtual tokens exactly the same applies as mentioned in 7.2.2.3.

7.2.4. C_InitToken

Not supported for SSCD applications – only for PKCS#11 generic application.

There are the following differences to the PKCS#11 specification:

- C_InitToken cannot be used to set an initial SO-PIN. Instead, the provided SO-PIN is used for authentication on the token. The factory default value is 16 ascii zeros (- hex-encoded):
  \x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30

Note:

The SO-PIN is 16-Byte long and is internally a 3DES-Key. 3-DES has a parity bit on the least significant digit which is not used for cryptography.

SO-PINs that only differ in this parity bit are the same.

This means the SO-PIN (hex-encoded):
\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30\x30

is the same as
\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31\x31
because it only differs in the last bit.

- The label-parameter of C_InitToken is not used for the label of the token. Instead it is used for the serialNumber of the token. The provided label-parameter must have exactly 16 bytes.

- The label (=serialNumber) is only set if the given slot is the primarySlot (- it is ignored for virtual slots).

7.2.5. C_InitPIN

C_InitPIN is used to unlock the corresponding PIN of the token.

Before C_InitPIN, a C_Login with the SO-PIN must be carried out.
Note: For SSCD applications, the SO-PIN is the so-called signature PUK.

For PKCS#11 generic applications, a new signature PIN can be passed in the "pPin" parameter of C_InitPIN.

For SSCD applications, the signature PIN can only be unlocked. However, a new signature PIN cannot be set. The "pPin" parameter of C_InitPIN must therefore be NULL and ulPinLen must be „0“.

7.2.6. C_GenerateRandom

C_GenerateRandom only supports the generation of random numbers in lengths 0 - 16

- If 0 is provided, an 8-byte random number is generated
- For 1-16, a random number of appropriate length is generated
- For > 16, CKR_DEVICE_ERROR is returned

7.2.7. C_SetAttributeValue

The function C_SetAttributeValue allows changing PKCS#11 attributes. These values are only changed in memory but not permanently written to the PKCS#11 token.

The reason for this is because there are calling applications that want to change certain PKCS#11 attributes and will not work if C_SetAttributeValue is not supported.

However, we strongly recommend not using this function as it involves a large risk - for example if CKA_IDs are changed.

Note:

In order to avoid irreparable damage, these settings are not permanently written to the card.

Changing the CKA_Label is generally not supported - not even in memory.

7.2.8. C_Sign

Each key length supports hash-lengths that are longer than the size of the key.
This means:

- An ECC-Key P256 also supports signing of hash values calculated by SHA-384 and SHA-512.
- An ECC-Key P384 also supports signing of hash values calculated by SHA-512.

This is supported by truncating the hash values to the key length.

7.2.9. C_GetMechanismList

The following applies specifically to SSCD tokens:

With SSCD there is exactly one SSCD key per virtual slot/token. This can be an RSA key or an ECC key.

However, the PKCS#11 module only knows whether it is RSA or ECC after a PIN verification (C_Login). This means that when logged out, all possible algorithms are returned - i.e. RSA and ECC.

In difference to the logged-in state, where all ECC algorithms are returned if it is an ECC key and all RSA algorithms if it is an RSA key

7.2.10. Authentication state

With C_Login of type CKU_USER, the signature PIN of the corresponding token is verified. As a result, commands can be executed that require successful PIN verification.

For PKCS#11 generic applications you are authenticated until a C_Logout is performed. This resets the authentication state of the signature PIN.

This is different with SSCD applications where the signature PIN is “consumed”. Ie if an operation is carried out after C_Login that requires PIN verification (e.g. C_Sign, C_CreateObject, etc) then this operation automatically resets the authentication state of the signature PIN. After the command you are no longer authenticated. A C_Login must therefore be carried out for every operation that requires the signature PIN.

In addition, there are the following special cases:

1)

If there are several applications on the ACDIS smart card and you change the application, the authentication state is automatically reset.

Example:

Assuming you select the token ACDIS.SSCD1.SIG1-active and perform a C_Login here. Then the authentication state of the associated signature PIN is set. Then you select the token of another SSCD application, namely ACDIS.SSCD2.SIG1-active, and also perform a C_Login here. Because the underlying smart card has two different applications (SSCD1 and SSCD2), the smart card applications change.

This means that the authentication state of the first PIN is lost and you have to do a C_Login again before you can use the assigned signature key.

2)

If you perform a C_Logout on a token (e.g.: ACDIS.SSCD1.SIG1-active), not only is the authentication state of the affected signature PIN reset, but also the authentication state of all signature PINs that were previously verified using C_Login.

7.2.11. Common pitfalls

Many PKCS#11 functions make use of extended-length APDUs.
If this is not supported by your card reader the function call returns error CKR_FUNCTION_REJECTED (= 0x200).
The most common cause of error is that the default USBCCID driver is used instead of the reader driver from the manufacturer. To solve the problem please install the manufacturer's drivers.

8. EXAMPLES

8.1. ACTIVATION OF SSCD SIGNATURE PIN

C_SetPIN must be called for this token. Where the transport PIN must be passed as the first parameter and the desired signature PIN as the second parameter.

Source-Code example:

```
p11->C_OpenSession(slotid, CKF_RW_SESSION | CKF_SERIAL_SESSION, NULL, NULL, &session);
p11->C_SetPIN(session, (CK_UTF8CHAR_PTR)"12345", 5, (CK_UTF8CHAR_PTR)"123456", 6);
p11->C_CloseSession(session);
```

8.2. READING OF SSCD PUBLIC-KEY

Reading the SSCD public keys is protected with the assigned signature PIN.

Source-Code example:

```
CK_BYTE modulus[512];
CK_BYTE exponent[3];

CK_OBJECT_CLASS classpuk = CKO_PUBLIC_KEY;
CK_ATTRIBUTE keytemplate[] = {
	{ CKA_CLASS, &classpuk, sizeof(classpuk) }
};
CK_ATTRIBUTE datatemplate[] = {
	{ CKA_MODULUS, &modulus, sizeof(modulus) },
	{ CKA_PUBLIC_EXPONENT, &exponent, sizeof(exponent) }
};

CK_OBJECT_HANDLE hnd;
CK_ULONG cnt = 0;

p11->C_OpenSession(slotid, CKF_RW_SESSION | CKF_SERIAL_SESSION, NULL, NULL, &session);

p11->C_Login(session, CKU_USER, (CK_UTF8CHAR_PTR)"123456", 6);

p11->C_FindObjectsInit(session, &keytemplate, sizeof(keytemplate) / sizeof(CK_ATTRIBUTE));

p11->C_FindObjects(session, &hnd, 1, &cnt);

p11->C_FindObjectsFinal(session);

if (cnt > 0) {
	p11->C_GetAttributeValue(session, hnd, (CK_ATTRIBUTE_PTR)&datatemplate, 2);

	unsigned char* modulus = datatemplate[0].pValue;
	int modulus_len = datatemplate[0].ulValueLen;

	unsigned char* exponent = datatemplate[1].pValue;
	int exponent_len = datatemplate[1].ulValueLen;

}

p11->C_Logout(session);

p11->C_CloseSession(session);
```

8.3. WRITE SSCD SIGNATURE CERTIFICATES

Writing SSCD signature certificates is protected with the assigned signature PIN.

Source-Code example:

```
CK_OBJECT_CLASS certClass = CKO_CERTIFICATE;

CK_ATTRIBUTE certTemplate[2];
int certAttributes = 0;
CK_OBJECT_HANDLE hndCert;

unsigned char* value;
int bytes_read;

// setting value and bytes_read to certificate data
// value = ...
// bytes_read = ...


p11->C_OpenSession(slotid, CKF_RW_SESSION | CKF_SERIAL_SESSION, NULL, NULL, &session);

p11->C_Login(session, CKU_USER, (CK_UTF8CHAR_PTR)"123456", 6);

certTemplate[certAttributes].type = CKA_CLASS;
certTemplate[certAttributes].pValue = &certClass;
certTemplate[certAttributes].ulValueLen = sizeof(certClass);
certAttributes++;

certTemplate[certAttributes].type = CKA_VALUE;
certTemplate[certAttributes].pValue = value;
certTemplate[certAttributes].ulValueLen = bytes_read;
certAttributes++;

rc = p11->C_CreateObject(session, certTemplate, certAttributes, &hndCert);

p11->C_Logout(session);

p11->C_CloseSession(session);
```

8.4. CREATION OF QUALIFIED SIGNATURES

Source-Code example:

```
char* tbs = "Hello World";
CK_BYTE signature[512];
CK_ULONG len=0;

CK_OBJECT_HANDLE hnd;
CK_ULONG cnt = 0;
CK_BBOOL _true = CK_TRUE;
CK_MECHANISM mech = { CKM_SHA256_RSA_PKCS_PSS, 0, 0 };
CK_RSA_PKCS_PSS_PARAMS pssparams = { CKM_SHA256, CKG_MGF1_SHA256, 32 };

CK_OBJECT_CLASS classprk = CKO_PRIVATE_KEY;
CK_ATTRIBUTE keytemplate[] = {
              { CKA_CLASS, &classprk, sizeof(classprk) },
              { CKA_SIGN, &_true, sizeof(_true) }
};

p11->C_OpenSession(slotid, CKF_RW_SESSION | CKF_SERIAL_SESSION, NULL, NULL, &session);

p11->C_Login(session, CKU_USER, (CK_UTF8CHAR_PTR)"123456", 6);

p11->C_FindObjectsInit(session, &keytemplate, sizeof(keytemplate) / sizeof(CK_ATTRIBUTE));

p11->C_FindObjects(session, &hnd, 1, &cnt);

p11->C_FindObjectsFinal(session);

if (cnt > 0) {
       p11->C_SignInit(session, &mech, hnd);

       // Get-Length of signature
       rc = p11->C_Sign(session, (CK_BYTE_PTR)tbs, (CK_ULONG)strlen(tbs), NULL, &len);

       // Sign Data
       p11->C_Sign(session, (CK_BYTE_PTR)tbs, (CK_ULONG)strlen(tbs), signature, &len);
}

p11->C_Logout(session);

p11->C_CloseSession(session);
```

9. TROUBLESHOOTING

9.1. LOG-FILES

The PKCS#11 library generates log files. The location depends on the operating system.

Windows:
The log files are located in the current user's temp directory:
C:\Users\<<current user>>\AppData\Local\Temp

Linux and macOS:
The log files are created in the HOME directory of the current user and here in the subdirectory "acdispkcs11". The PKCS#11 library expects that the subdirectory exists, ie it must be created manually - HOME/acdispkcs11

The file names of the log files are: acdispkcs11_0.txt, acdismini_1.txt, etc.
The last digit of the file name corresponds to the ones digit of the current date.

Example:
Let's assume that today is September 5th, 2023. The ones digit of the current day is therefore "5". So the PKCS#11 library writes log statements to the file acdispkcs11_5.txt. Log statements of the respective day are appended to the file. However, if the file is older, it will be overwritten.
In total, this mechanism leads to a maximum of 10 log files: acdispkcs11_0.txt - acdispkcs11_9.txt

10. COMMON TOOLS

10.1. OPENSC PKCS#11-TOOL

OpenSC's pkcs11-tool is a very common tool for calling PKCS#11 functions from the command line. Some example calls are given below:

Windows (64-Bit):

Function	Usage
List Slots	"C:\Program Files\OpenSC Project\OpenSC\tools\pkcs11-tool" --modul acdis-pkcs11-64.dll --list-slots
List Objects	Certificates: "C:\Program Files\OpenSC Project\OpenSC\tools\pkcs11-tool" --modul acdis-pkcs11-64.dll --list-objects --type cert Private keys: "C:\Program Files\OpenSC Project\OpenSC\tools\pkcs11-tool" --modul acdis-pkcs11-64.dll --login --login-type user --list-objects --type privkey
Get Version	"C:\Program Files\OpenSC Project\OpenSC\tools\pkcs11-tool" --modul acdis-pkcs11-64.dll --show-info

Linux:

Function	Usage
List Slots	pkcs11-tool --modul libacdis-pkcs11.so --list-slots
List Objects	Certificates: pkcs11-tool --modul libacdis-pkcs11.so --list-objects --type cert Private Keys:

	pkcs11-tool --modul libacdis-pkcs11.so --login --login-type user --list-objects --type privkey
Get Version	pkcs11-tool --modul libacdis-pkcs11.so --show-info

macOS:

Function	Usage
List Slots	pkcs11-tool --modul acdis-pkcs11.dylib --list-slots
List Objects	Certificates: pkcs11-tool --modul acdis-pkcs11.dylib --list-objects --type cert Private Keys: pkcs11-tool --modul acdis-pkcs11.dylib --login --login-type user --list-objects --type privkey
Get Version	pkcs11-tool --modul acdis-pkcs11.dylib --show-info

10.2. ACROBAT READER

Acrobat generally supports PKCS#11, but with some restrictions:

- Only RSA PKCS#1v1.5 is supported. Support for RSA-PSS is not currently implemented by Acrobat Reader.
- ECDSA is also not currently implemented by Acrobat Reader.

For Acrobat Reader under Windows, we therefore recommend using the ACDIS Minidriver instead of the PKCS#11 library.

Please consult minidriver documentation for details.

There are the following to note:

If the PKCS#11 library and the Minidriver are configured in parallel in Acrobat Reader and you create a signature via Minidriver, you will see an invalid signature.

If you then click on the signature details then a PKCS#11 error is displayed.
This is a bug in Acrobat Reader. Although signing is done via Minidriver, Acrobat does something on the PKCS#11 module.
However, the signature is not invalid. This means that if you close the document and reopen it, the signature verification works.

We therefore recommend signing in Acrobat Reader exclusively via ACDIS Minidriver. The ACDIS PKCS#11 library should not be configured to avoid the error described above.

11. APPENDIX

11.1. THIRD-PARTY COPYRIGHTS

11.1.1. OpenSSL

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)
The OpenSSL toolkit stays under a double license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit.

```
OpenSSL License
---------------

/* ====================================================================
 * Copyright (c) 1998-2019 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

```
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
```

```
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledge-
ment:
 *    "This product includes software written by Tim Hudson
(tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

11.1.2. sc-hsm-embedded

The PKCS#11 library uses source code from sc-hsm-embedded project. Sc-hsm-embedded is Copyright (c) 2013, CardContact Systems GmbH, Minden, Germany and licensed under 3-clause BSD license.