Atos Worldline

Haachtsesteenweg 1442
1130 Brussels
Belgium

DEP Documentation

DEP PKCS#11 User Guide

Version: 04.07 Classification: Public

Version Management Report

Version	**Name(s)**	**Date**	**Comment**
04.00	Anna Papayan	02.02.2011	First version
04.01	Anna Papayan	17/05/2011	Update by including the description of install packages and the installing procedure for Windows and Linux.
04.02	Anna Papayan	06/06/2011	Add OpenSSL/DEP PKCS#11 integration.
04.03	Anna Papayan	27/07/2011	New functionality added, crypto.cfg updated for OpenSSL
04.04	Joris Delclef	04/08/2011	Add EJBCA information
04.05	Anna Papayan	05/08/2011	Minor changes
04.06	Anna Papayan	02/09/2011	timeout_connect parameter description added. Certificate object added in the *Logical View of DEP Token* chapter.
04.07	Anna Papayan	26.04.2012	Minor changes

1. TABLE OF CONTENTS

2. SCOPE OF THE DOCUMENT

This document introduces the DEP PKCS#11 API library for Banksys DEP Hardware Security Modules (HSM) and provides information about the installation, configuration use and integration of this library.

It gives an overview of:

- Supported functions;
- Supported operating systems;
- Installation and configuration of DEP PKCS#11 library;
- Load balancing mechanism implemented in DEP PKCS#11 library;
- Key management behaviour in load balancing environment.

The installation and configuration phase details how to install and configure the DEP PKCS#11 library on each supported operating system.

At the load balancing phase the document details what is a load balancing, describes the error mechanisms, failover procedures in load balancing environment and key management behaviour in load balancing environment.

This guide is intended for software developers and Customer Security Officers.

2.1. REFERENCES

This document contains references to other documents. This paragraph gives a list of all the documents referred to.

- *PKCS #11 Base Functionality v2.20*
- *PKCS#11 v2.20: Cryptographic Token Interface Standard*
- *PKCS#11 Mechanisms v2.30*

PKCS#11 documents and information are available online at `http://www.rsasecurity.com/rsalabs/PKCS/node.asp?id=2133`.

More information about the OpenSSL crypto library and the distribution tarballs can be found online at `www.openssl.org`. For the information about the OpenSC project refer to the `www.opensc-project.org` web page.

There are no references made to the following documents, but they could be useful to understand this document.

- *DEP Introduction to DEP*
- *DEP Glossary*

2.2. CONTACTING ATOS WORLDLINE

You can visit *Atos Worldline* on the World Wide Web to find out about new products and about various other fields of interest.
URL: www.atosworldline.com.

For the documentation visit www.banksys.com web page.

For support on issues related to DEP, customers, partners, resellers, and distributors can send an email to the DEP Hotline:
mailto: dephotline-atosworldline@atosorigin.com.

3. INTRODUCTION

The DEP PKCS#11 library is a cryptographic library which manages standardized access to DEP Crypto Modules.

PKCS#11 standard defines a platform independent API to cryptographic devices. The specified API is called “Cryptoki”, which is stands for a “Cryptographic Token Interface“. Cryptoki is an Application Programming Interface (API) with devices that hold cryptographic information and perform cryptographic functions. Cryptoki follows a simple object-based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a “cryptographic token”.

For more information about Cryptographic Token Interface standard refer to the *RSA standard PKCS#11 v2.20: Cryptographic Token Interface Standard* document.

The DEP PKCS#11 library is an Atos Worldline’s implementation of Cryptoki, that gives cryptographic applications standardized access to the DEP Platforms and DEP Crypto Modules.

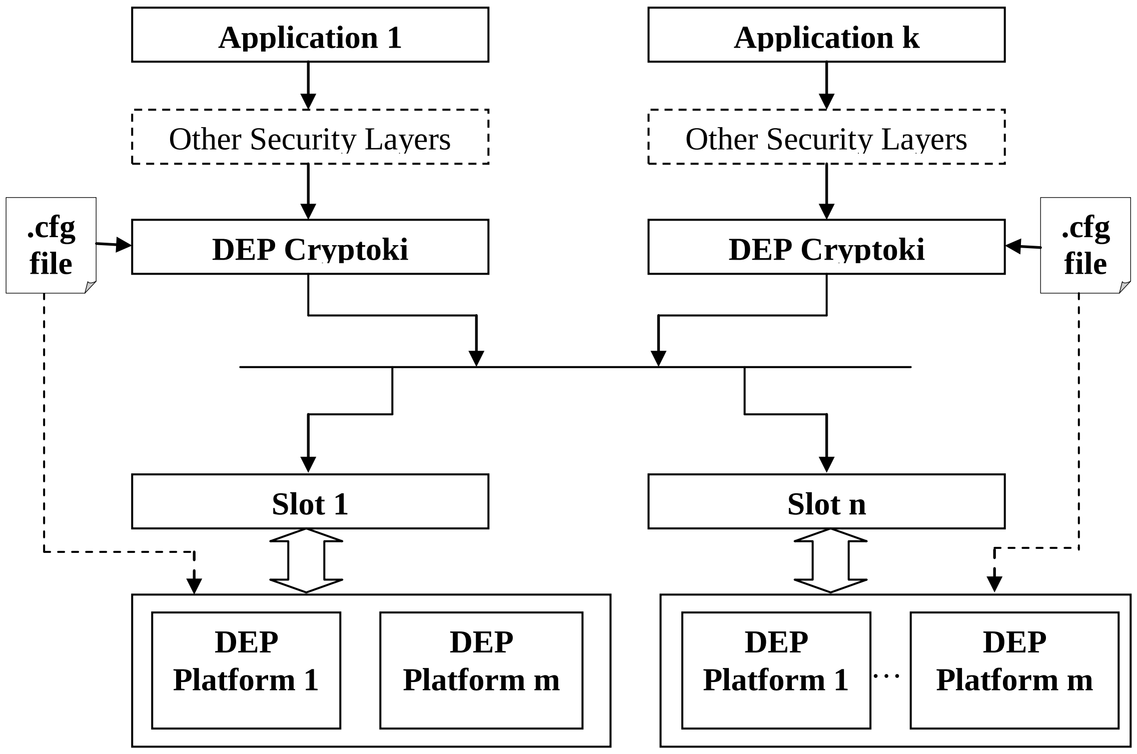
The DEP PKCS#11 library is compliant with the *PKCS#11 v2.20* standard. For more information about PKCS#11 standard refer to the *PKCS #11 Base Functionality v2.20.*

Application can access the DEP PKCS#11 library in multithreaded or mono threaded fashion.

4. MAPPING STANDARD/LIBRARY

4.1. DEP CRYPTOKI MODEL

DEP Cryptoki is intended to be an interface between applications and DEP Platforms. DEP Cryptoki's general model is illustrated below:

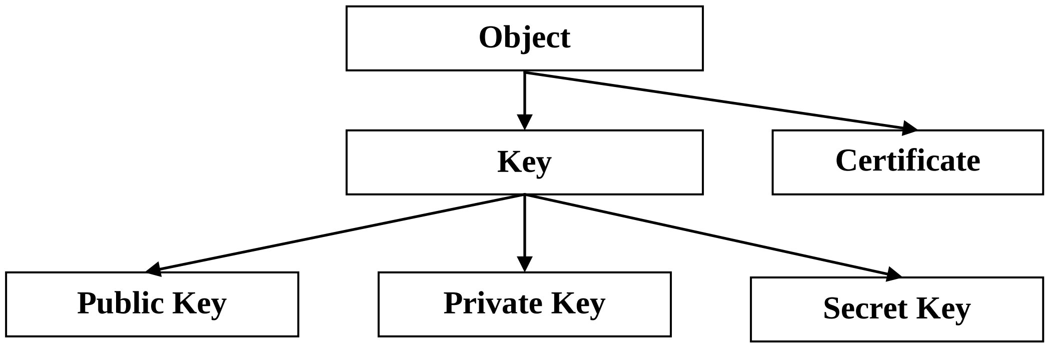


Cryptoki provides an interface to one or more DEP Platforms/DEP Crypto Modules that are active in the system through number of "slots", which configurations are given in DEP PKCS#11 library configuration file. DEP Platforms with the same configuration can be present in different slots.

4.1.1. Logical view of DEP Token

Cryptoki's logical view of a Token is a device that stores objects and can perform cryptographic functions. Cryptoki defines three classes of object: data, certificates, and keys.

DEP Cryptoki supports the Key and Certificate classes. The Key class stores the cryptographic keys. The key may be a public key, a private key, or a secret key. The Certificate class stores the certificate. This view is illustrated in the following figure:



An object consists of a set of attributes, each of which has a given value. Each attribute that an object possesses has precisely one value. The DEP PKCS#11 library only deals with the key and certificate objects attributes specified in 2.20 version of the RSA Standard.

4.1.2. Key Objects

The DEP PKCS#11 library supports only the key object which holds encryption and authentication keys, which can be private keys or secret keys; each of these types of keys has subtypes for use in specific cryptographic operations. DEP Cryptoki uses the RSA private key objects, which hold the RSA private keys, and the AES Secret Key objects which hold AES keys.
For more information about key objects, their subtypes and object attributes refer to the documents *PKCS#11 Base Functionality v2.20* and *PKCS#11 Mechanisms v2.30*.

4.1.2.1.RSA Private Key object attributes

DEP Cryptoki attributes for RSA private keys	Description	Value
CKA_ID	Key tag in DEP Crypto Module	04 25 07 xx
CKA_LABEL	Key label	K_PKI_RSA_PRIV_KEY
CKA_MODULUS	Modulus	The value is retrieved from DEP Crypto Module
CKA_PUBLIC_EXPONENT	Public exponent	The value is retrieved from DEP Crypto Module

4.1.2.2.AES Secret Key object attributes

DEP Cryptoki attributes for AES secret keys	Description	Value
CKA_ID	Key tag in DEP Crypto Module	04 31 04 xx
CKA_LABEL	Key label	K_SCRYP_AES

4.2. USERS SUPPORTED IN DEP PKCS#11 LIBRARY

The DEP PKCS#11 library doesn’t support the Security Officer role and only the normal user role is supported. Only the normal user is allowed to perform cryptographic operations and to access to private objects on the token, and that access is granted only after the normal user has been authenticated.

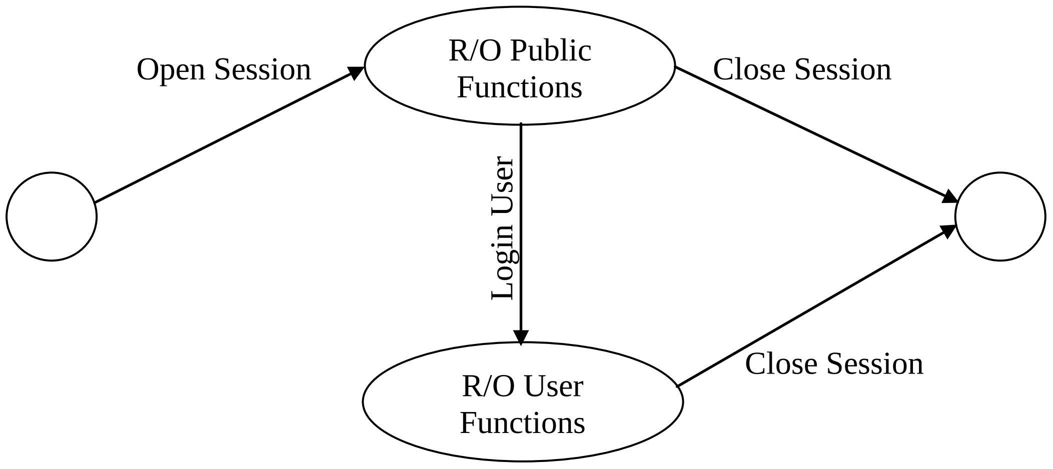
For authentication the application must pass the ‘D’ ‘E’ ‘P’ sequence of ASCII symbols.

4.3. SESSIONS

Cryptoki requires that an application open one or more sessions with a DEP Token to gain access to the Token's objects and functions. A session provides a logical connection between the application and the Token. A session in general can be a read/write (R/W) session and a read-only (R/O) session.

DEP Cryptoki supports both R/O and R/W session types.

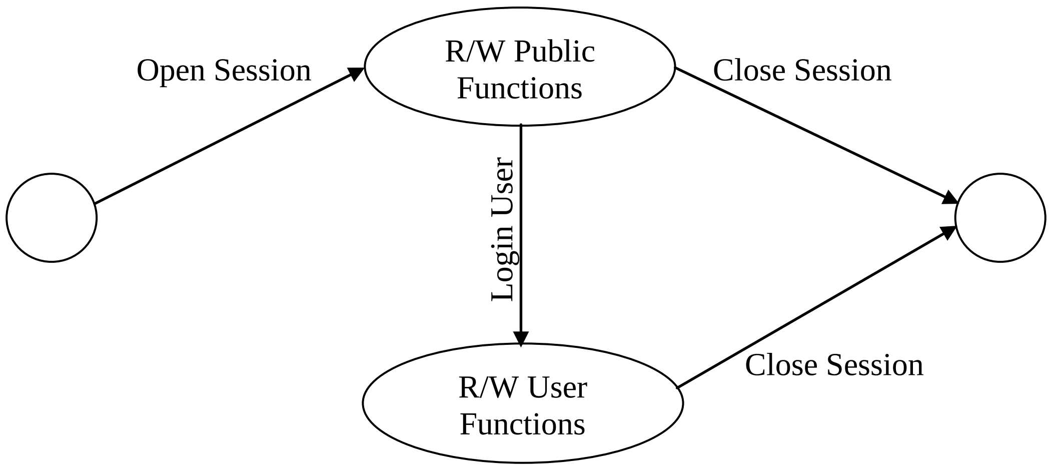
A read-only (R/O) session can be in one of two states, as illustrated in the figure below. When the session is initially opened, it is in either the “R/O Public Session” state or the “R/O User Functions” state.



The following table describes the session states supported in DEP PKCS#11 library.

State	Description
R/O Public Functions	The application has opened a read-only session. The application has read-only access to public token objects and read/write access to public session objects.
R/O User Functions	The normal user has been authenticated to the token. The application has read-only access to all token objects (public or private) and read/write access to all session objects (public or private).

A read/write (R/W) session, like an R/O session type, can be in one of two states. When the session is opened, it is in either the “R/W Public Functions” state or the “R/W User Functions” state.



The following table describes the session states supported in DEP PKCS#11 library.

State	Description
R/W Public Functions	The application has opened a read/write session. The application has read/write access to all public objects.
R/W User Functions	The normal user has been authenticated to the token. The application has read/write access to all objects.

Multiple sessions on multiple tokens are supported by DEP Cryptoki. An application may have one or more sessions with one or more DEP Tokens. In general, a DEP Token may have multiple sessions with one or more applications. A particular Cryptoki slot may have maximum 128 read-only or read/write sessions.
After opening a session the application has access to the DEP Token’s public and private objects.

Objects that reside on the token are referred to as token objects. Objects that exist only for the duration of a session are referred to as session objects. When a session is closed, any session objects which were created in that session are destroyed. This holds even for session objects which are being used in other sessions. That is, if a single application has multiple sessions opened with a token, and it uses one of them to create a session object, then that session object is visible through any of that application's sessions. However, as soon as the session that was used to create the object is closed, that object is destroyed.

For more information about sessions and session states refer to the *PKCS #11 Base Functionality v2.20.*

4.4. FUNCTIONS SUPPORTED BY DEP CRYPTOKI

The DEP Cryptoki consists of number of functions, slot and token management, object management and cryptographic functions. Not all the functions specified in PKCS#11 v2.20 are currently implemented in DEP PKCS#11 library. This chapter indicates the functions that are implemented in DEP PKCS#11 library.

For more information about all functions, their return values and mechanisms refer to the documents *PKCS #11 Base Functionality v2.20* and *PKCS#11 Mechanisms v2.30.*

4.4.1. General purpose functions

Function	Description
C_Initialize	initializes the Cryptoki library
C_Finalize	cleans up miscellaneous Cryptoki-associated resources
C_GetFunctionList	obtains all entry points (i.e. functions) of the Cryptoki library

4.4.2. Slot and Token Management functions

Function	Description
C_GetSlotList	obtains a list of slots from the library
C_GetSlotInfo	obtains general information about Slot.
C_GetTokenInfo	obtains general information about Token
C_GetMechanismList	obtains a list of mechanism types supported by a Token

4.4.3. Session Management functions

Function	Description
C_OpenSession	opens a session between an application and a particular token
C_CloseSession	closes a session
C_Login	logs into a token
C_GetSessionInfo	obtains information about a session

4.4.4. Object Management functions

Function	Description
C_GetAttributeValue	obtains the value of an attribute
C_FindObjectsInit	initializes an object search operation
C_FindObjects	continues an object search operation
C_FindObjectsFinal	finishes an object search operation
C_CreateObject[1]	creates a new object. The DEP PKCS#11 library supports the creation of X.509 certificate objects (certificate type CKC_X_509) of CKO_CERTIFICATE object class.
C_SetAttributeValue	modifies the value of one or more attributes of an object

4.4.5. Cryptographic functions

Category	Function	Description
Encryption functions	C_EncryptInit	initializes an encryption operation
	C_Encrypt	encrypts single-part data. The CKM_AES_CBC mechanism is supported by this operation.
	C_EncryptUpdate	continues a multiple-part encryption operation
	C_EncryptFinal	finishes a multiple-part encryption operation
Decryption functions	C_DecryptInit	initializes a decryption operation
	C_Decrypt	decrypts single-part encrypted data. The CKM_AES_CBC and CKM_RSA_PKCS mechanisms are supported by this operation.
	C_DecryptUpdate	continues a multiple-part decryption operation
	C_DecryptFinal	finishes a multiple-part decryption

[1] Only Token objects will be created by this function. The created certificate objects will be stored in **certificate** folder under the **Storage** path, which will contain appropriate <*host_token number*> subfolders with certificate objects in .xml format. Note that to use the created certificate objects from another host (in case of the same configuration), all the content of **Storage** folder must be copied to the destination host.
Not supported in load balancing configurations. Otherwise, unexpected behaviour may occur.

		operation
Signing functions	C_SignInit	initializes a signature operation
	C_Sign	signs single-part data. The CKM_RSA_PKCS mechanism is supported by this operation.
	C_SignUpdate	continues a multiple-part signature operation, processing another data part. The CKM_SHA1_RSA_PKCS mechanism is supported by this operation.
	C_SignFinal	finishes a multiple-part signature operation, returning the signature. The CKM_SHA1_RSA_PKCS mechanism is supported by this operation.

4.4.6. Key Management functions

Function	Description
C_GenerateKeyPair[2]	generates a public/private key pair, creating new key objects. The CKM_RSA_PKCS_KEY_PAIR_GEN mechanism is supported by this operation.

[2] Not supported in load balancing configurations. Otherwise, unexpected behaviour may occur. Only Token objects will be created by this function.

5. INSTALLING DEP PKCS#11

The purpose of this chapter is to describe the DEP PKCS#11 library installation process.

The DEP PKCS#11 library currently works with all recent 32-bit and 64-bit versions of Windows and also with Linux environments.

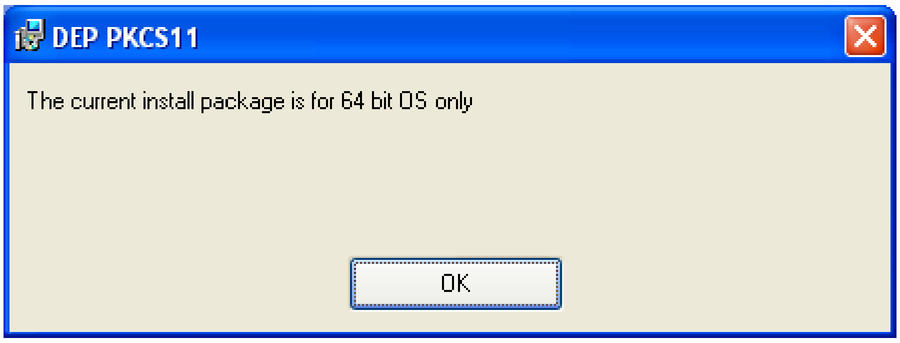
Two different procedures are described below; the installation for MS Windows environment and for Linux environment.

5.1. INSTALL ON MS WINDOWS ENVIRONMENT

An installation procedure is available for the DEP PKCS#11 library. It is a wizard-driven procedure that lets you to install the DEP PKCS#11 library on your computer.

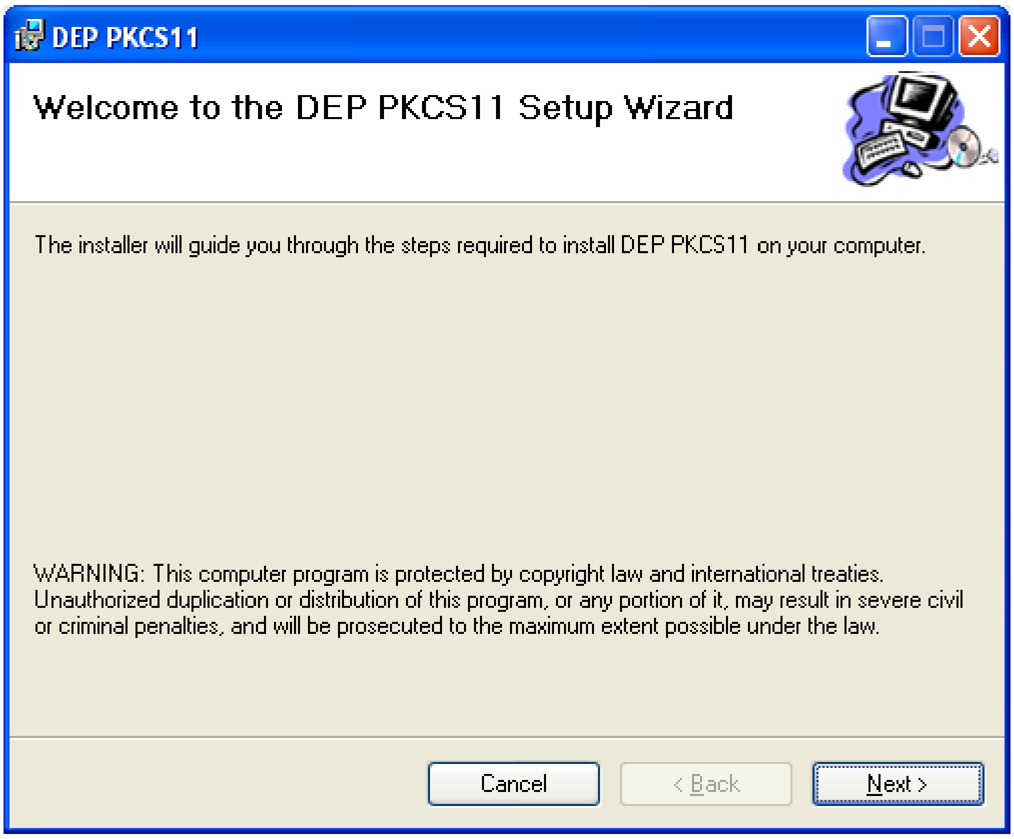
Extract the delivered archive (.zip) in any directory. There are two installation packages: **installerx64** for 64-bit version of Windows and **installerx86** for 32-bit version.

The following message-box will appear if you will try to install the DEP PKCS#11 library for 64-bit operating system on your 32-bit operating system.



The current install package is for 64 bit OS only

Choose the appropriate install package and run the **.msi** file. The **Welcome** dialog box appears and you can proceed with the installation by clicking **Next** and following the instructions that the wizard presents.

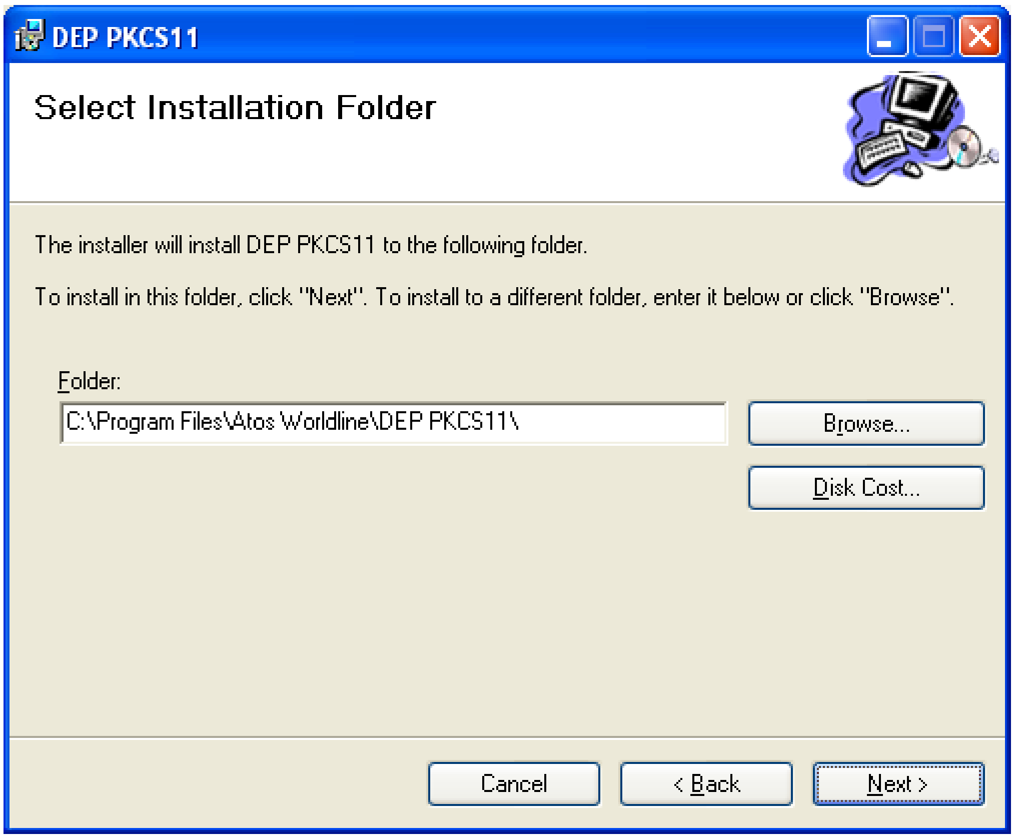


The installer will guide you through the steps required to install DEP PKCS11 on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

In the **Select Installation Folder** dialog box you have to specify the path to the folder where the DEP PKCS#11 library will be installed.
The default path is `C:\Program Files\Atos Worldline\ DEP PKCS11\`. It is recommended to use the default path, yet you can specify a different folder by clicking **Browse** and selecting the desired folder for the installation of the DEP PKCS#11 library.

Click **Next** to continue. If you want to abort the procedure, click **Cancel**.



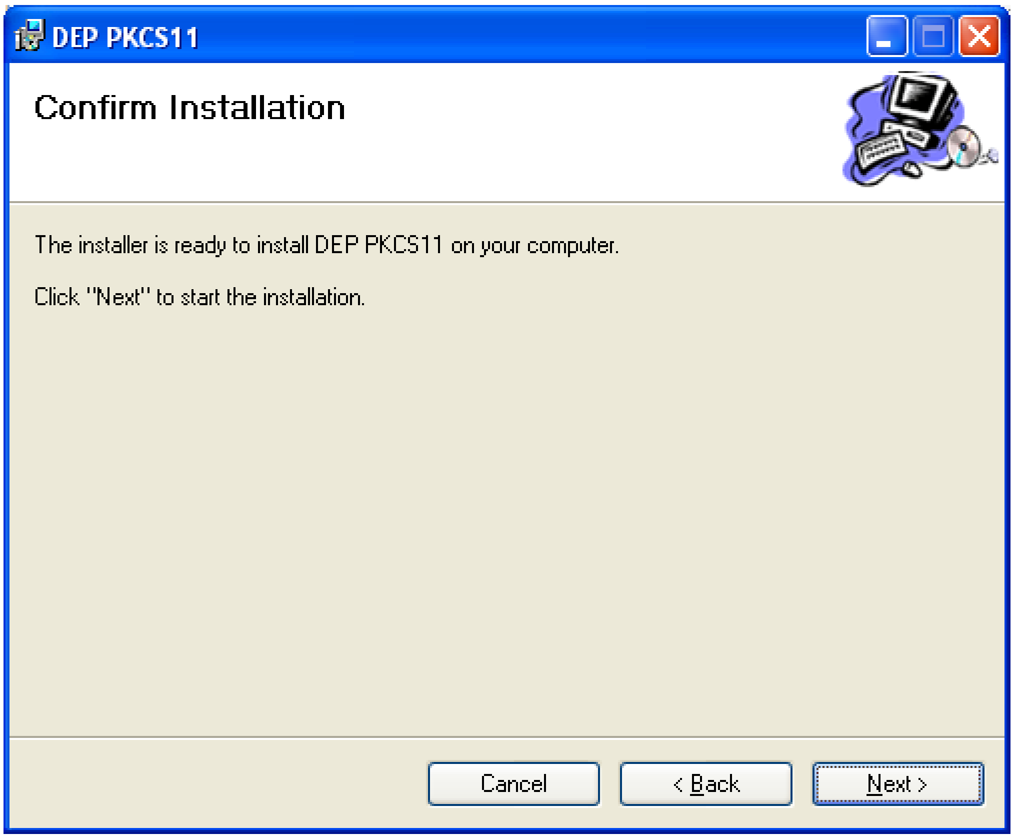
The installer will install DEP PKCS11 to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder:
C:\Program Files\Atos Worldline\DEP PKCS11\

The **Confirm Installation** dialog box that gives an overview of the settings selected during the installation procedure will appear.

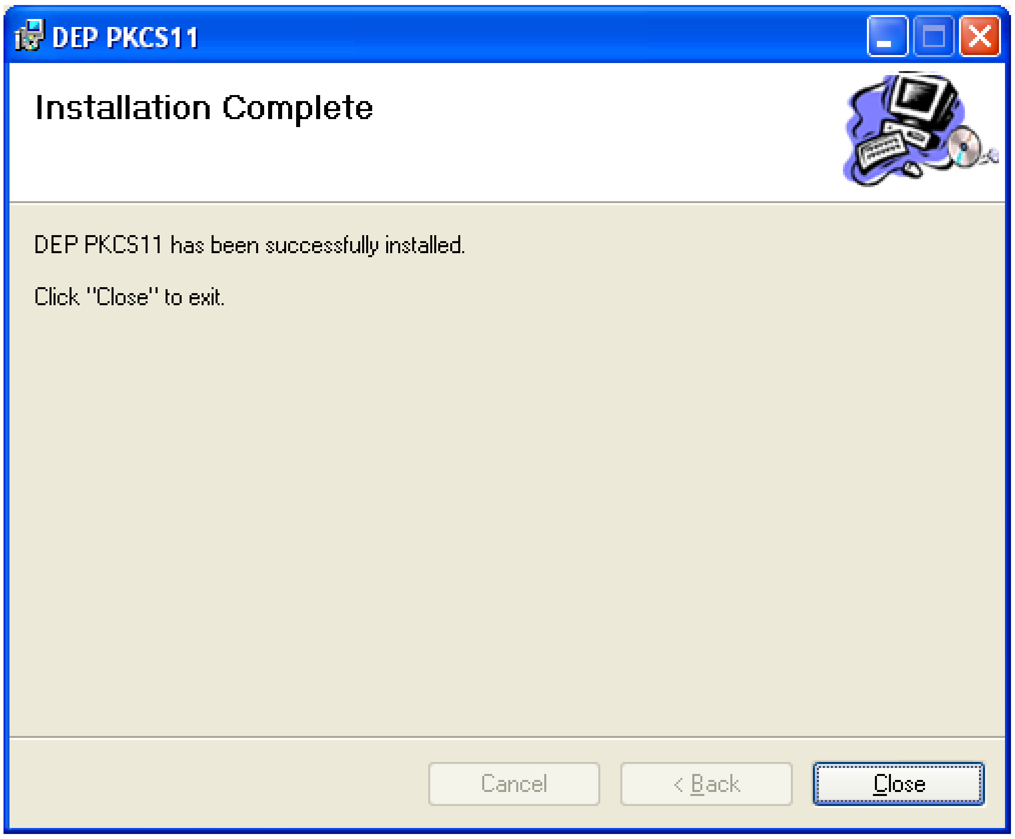
Click **Next** to continue. If you want to return to the previous screen, press **Back** or if you want to abort the procedure, press **Cancel**.



The installer is ready to install DEP PKCS11 on your computer.

Click "Next" to start the installation.

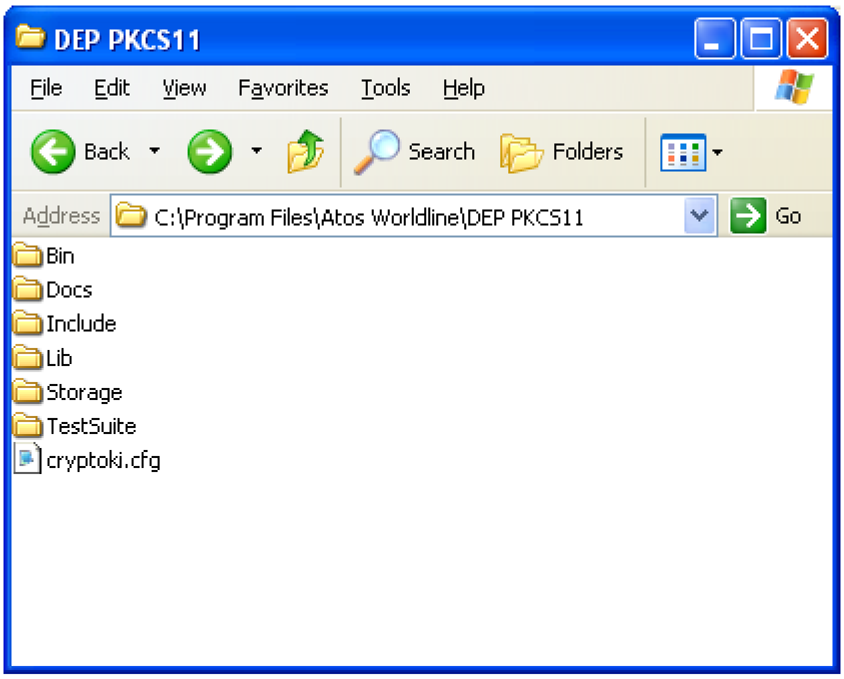
Once you have confirmed the installation options, the actual installation starts. A progress bar combined with status information show you how the installation moves on. After completing the installation procedure, the **Installation Complete** dialog box appears to notify you of a successful installation.
Click **Close** to exit.



DEP PKCS11 has been successfully installed.

Click "Close" to exit.

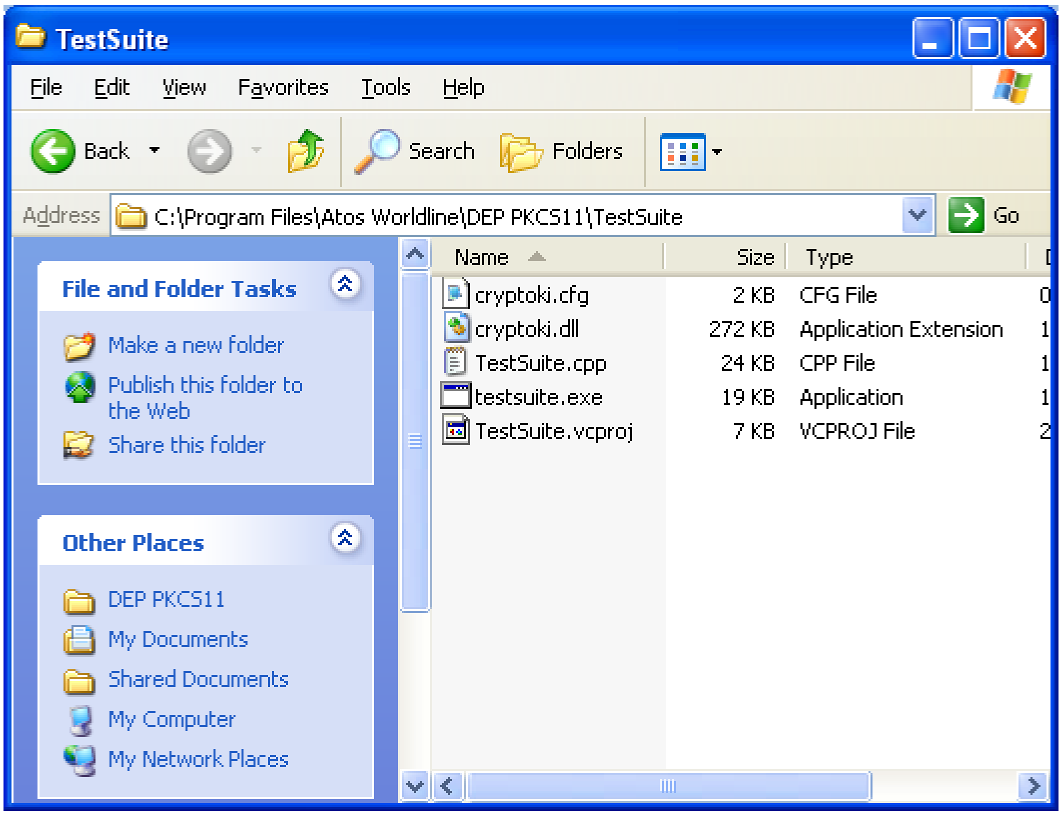
The installation setup creates the PKCS#11 files under the default path `C:\Program Files\Atos Worldline\ DEP PKCS11\`, or in the installation folder which you have specified during the installation. The PKCS#11 files are contained in the following directories:



C:\Program Files\Atos Worldline\DEP PKCS11

- Bin
- Docs
- Include
- Lib
- Storage
- TestSuite
- cryptoki.cfg

The **Bin** folder contains the *cryptoki.dll* file which should be found by the DEP Cryptoki application to able to use the DEP PKCS#11 services.
In the **Docs** folder you can find the documentation about the DEP PKCS#11 library.
The **Include** folder contains C header files (*.h)* that define the standard PKCS#11 declarations and can be included with Cryptoki application projects written on native C/C++ programming language.
The **Lib** folder contains the *cryptoki.lib* object library file for building the Cryptoki applications with the static library linkage.
The **Storage** folder contains the token objects created by Cryptoki application, but not supported by DEP (certificate objects).
And the **TestSuite** folder contains the following source tree to test the Cryptoki application:



Name	Size	Type
cryptoki.cfg	2 KB	CFG File
cryptoki.dll	272 KB	Application Extension
TestSuite.cpp	24 KB	CPP File
testsuite.exe	19 KB	Application
TestSuite.vcproj	7 KB	VCPROJ File

After preparing the *cryptoki.cfg* configuration file, run the *testsuite.exe* executable with the appropriate input arguments to start the specific test.
The *testsuite.exe* can be also produced by compiling the *TestSuite.vcproj* file with the MS Visual Studio 2008.

To use the DEP PKCS#11 library, the following steps should be done.

- Add the full path of *cryptoki.dll* file (`<installdirectory>\Bin`, where the *installdirectory* is the directory where the DEP PKCS#11 library was installed) to the system or user PATH environment variables, so the DEP cryptoki application can find the *cryptoki.dll* file,

- Or you can copy the *cryptoki.dll* file to the same directory as the DEP Cryptoki application or at any path defined by environment (e.g. `C:\WINDOWS\system32`), so that the DEP Cryptoki application can find it;
- Prepare the configuration file (see paragraph 6 on page 23) based on provided *cryptoki.cfg* template;
- Copy the *cryptoki.cfg* to the same directory as the *cryptoki.dll* or the DEP Cryptoki application. Note that first the DEP PKCS#11 library will look for the *cryptoki.cfg* file in the same directory as the *cryptoki.dll* file.

The DEP PKCS#11 library should be removed using the **Add/Remove Programs** utility in the Control Panel. This will remove all the DEP PKCS#11 files. After uninstalling the library, the `<installdirectory>\Bin` directory will not be removed from the system or user PATH environment variables. The PATH should be removed manually.

5.2. INSTALL ON LINUX ENVIRONMENT

The delivered PKCS#11 archive (.tar.gz) should be unpacked in any directory using either

```
tar xvf filename.tar.gz,
```

or

```
gzip -dc filename.tar.gz | tar x -
```

command.

You will have the **pkcs11_installer** folder.
To install the DEP PKCS#11 library, proceed with the following steps:

1. Run the `./configure` command to configure the PKCS#11 build environment,
2. If the configuration is successfully done, run the `make` command to compile the source codes. The *libdepp11.so* file will be generated under **pkcs11_installer** directory. The native GCC compiler can be used to create a *libdepp11.so* file. The build of DEP Cryptoki library uses only standard runtime libraries and does not require any additional third party libraries.
3. Run the `sudo make install` command to install the PKCS#11 library. This command will install the PKCS#11 library into the shared library location:

   ```
   /usr/lib/
   ```

To make the *cryptoki.cfg* configuration file accessible by the DEP PKCS#11 library, copy the *cryptoki.cfg* to the user's **home** folder or to the same directory as the DEP Cryptoki application. Note that first the DEP PKCS#11 library will look for the *cryptoki.cfg* file in user's **home** folder.

To uninstall the PKCS#11 library, run the `sudo make` command with the `uninstall` target.

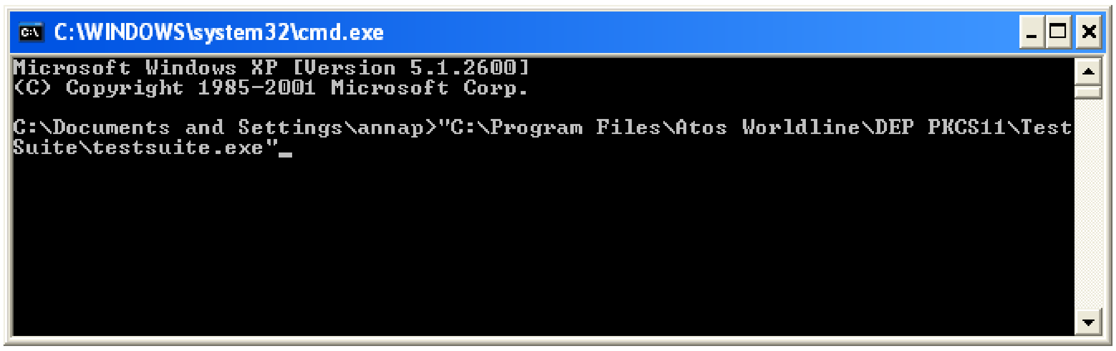
The PKCS#11 installer includes also the **testsuite** cryptoki testing application. This application must be built separately with the use of `make` command. After preparing the *cryptoki.cfg* file (see paragraph 6.1 on page 24), you can run the generated *testsuite* executable.

5.3. TESTING THE DEP PKCS#11 LIBRARY

The TestSuite application is designed to test the basic functionalities of newly installed DEP PKCS#11 library.

First the *cryptoki.cfg* file should be configured so that the PKCS#11 library can use it (see paragraph 6.1 on page 23).

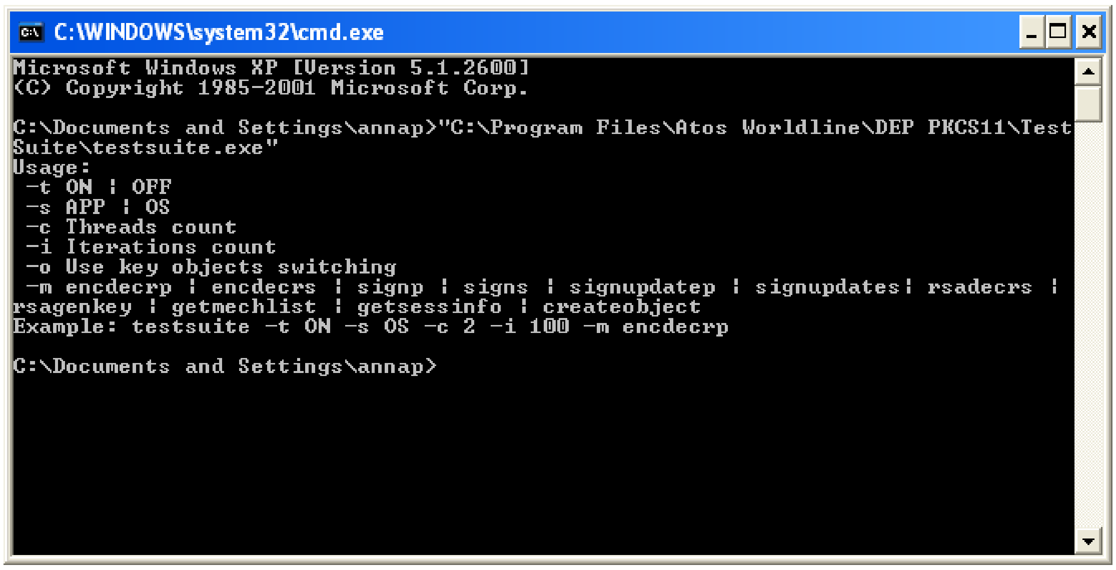
To execute the TestSuite application on Windows, run the *testsuite.exe* as follows:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\annap>"C:\Program Files\Atos Worldline\DEP PKCS11\Test
Suite\testsuite.exe"_
```

After the execution of *testsuite.exe*, you will have the list of input parameters.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\annap>"C:\Program Files\Atos Worldline\DEP PKCS11\Test
Suite\testsuite.exe"
Usage:
 -t ON | OFF
 -s APP | OS
 -c Threads count
 -i Iterations count
 -o Use key objects switching
 -m encdecrp | encdecrs | signp | signs | signupdatep | signupdates| rsadecrs |
rsagenkey | getmechlist | getsessinfo | createobject
Example: testsuite -t ON -s OS -c 2 -i 100 -m encdecrp

C:\Documents and Settings\annap>
```

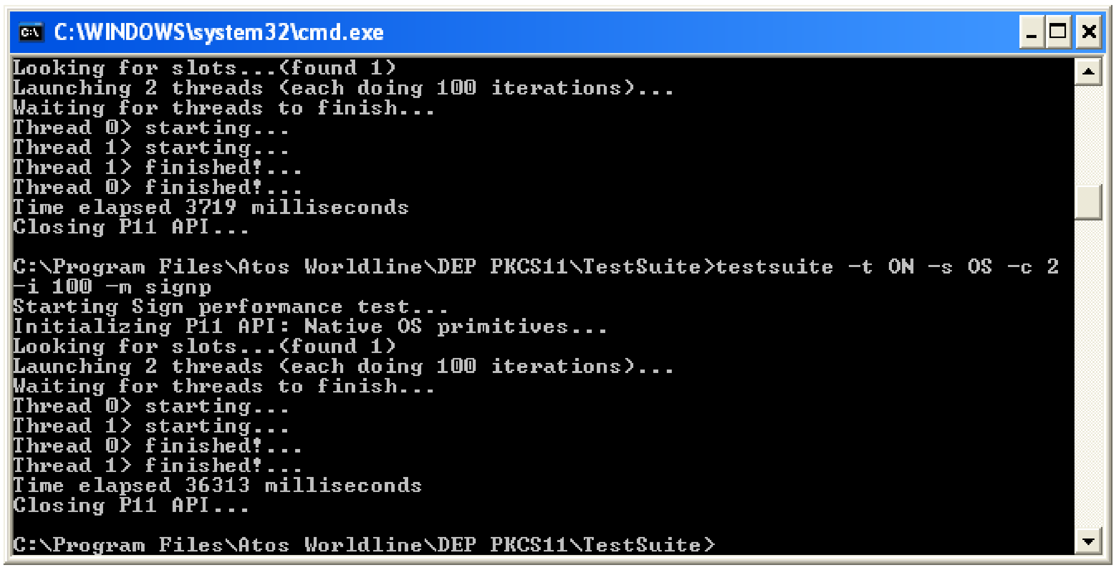
All the usage parameters are given in the following table:

Parameter	Value	Description
-t	ON OFF	Thread support on DEP PKCS#11 library is ON NO thread support on DEP PKCS#11 library
-s	APP OS	Synchronization primitives provided by the cryptoki application will be used Native operating system primitives will be used by the DEP PKCS#11 library
-c		Number of threads
-i		Number of iterations per thread
-o		Key object switching will be used after each test iteration
-m	encdecrp encdecrs signp signs signupdatep signupdates rsadecrs rsagenkey getmechlist getsessinfo createobject	Encryption/Decryption performance test Encryption/Decryption stress test Sign in single part performance test Sign in single part stress test Sign in multiple parts stress test Sign in multiple parts performance test RSA Decrypt stress test RSA Generate key pair test Getting mechanism list test Getting session info test Creating object test

Table 1

To create any test case, run the *testsuite* command with appropriate parameters.

Below is an example of a Sign in single part performance test that is run on Windows environment.



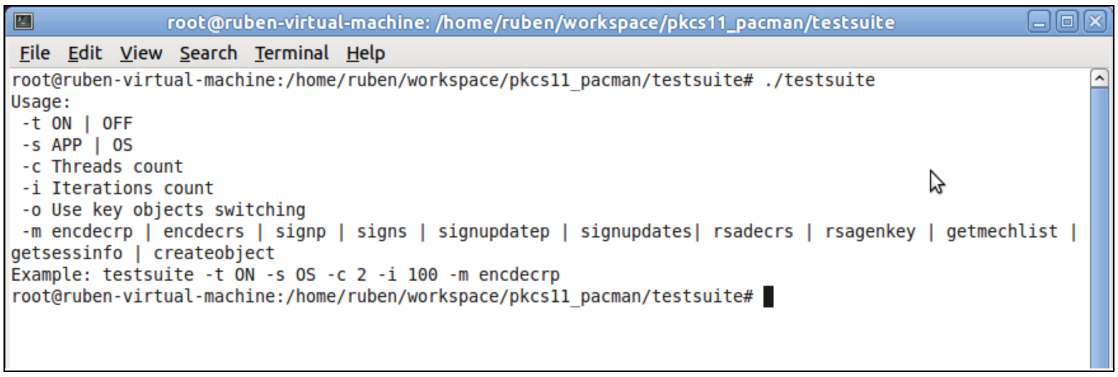
```
Looking for slots...(found 1)
Launching 2 threads (each doing 100 iterations)...
Waiting for threads to finish...
Thread 0> starting...
Thread 1> starting...
Thread 1> finished!...
Thread 0> finished!...
Time elapsed 3719 milliseconds
Closing P11 API...

C:\Program Files\Atos Worldline\DEP PKCS11\TestSuite>testsuite -t ON -s OS -c 2
-i 100 -m signp
Starting Sign performance test...
Initializing P11 API: Native OS primitives...
Looking for slots...(found 1)
Launching 2 threads (each doing 100 iterations)...
Waiting for threads to finish...
Thread 0> starting...
Thread 1> starting...
Thread 0> finished!...
Thread 1> finished!...
Time elapsed 36313 milliseconds
Closing P11 API...

C:\Program Files\Atos Worldline\DEP PKCS11\TestSuite>
```

The same principles are used when running the TestSuite application on Linux environment.

Run the generated testsuite executable using the *./testsuite* command, and create the appropriate test cases using the input arguments described in Table 1.


root@ruben-virtual-machine: /home/ruben/workspace/pkcs11_pacman/testsuite
File Edit View Search Terminal Help

```
root@ruben-virtual-machine:/home/ruben/workspace/pkcs11_pacman/testsuite# ./testsuite
Usage:
 -t ON | OFF
 -s APP | OS
 -c Threads count
 -i Iterations count
 -o Use key objects switching
 -m encdecrp | encdecrs | signp | signs | signupdatep | signupdates| rsadecrs | rsagenkey | getmechlist |
getsessinfo | createobject
Example: testsuite -t ON -s OS -c 2 -i 100 -m encdecrp
root@ruben-virtual-machine:/home/ruben/workspace/pkcs11_pacman/testsuite# 
```

6. CONFIGURING DEP PKCS#11

To use the DEP PKCS#11 library the following conditions should be true for DEP Platforms and the DEP Crypto Modules:

- The DEP Platform software should be VENUS 4.1.6 or higher, which supports 128 connections simultaneously;
- DEP Application Software should be loaded on DEP Crypto Modules. It should support the following interfaces:
  - **I_STD_GET_DEP_INFO,**
  - **I_STD_GET_TAG_STATUS**,
  - **I_PKI_OUTPUT_RSA_PUBLIC_FROM_KEY_TABLE,**
  - **I_STD_ECHO,**
  - **I_PKI_RSA_MODEXP_SK**,
  - **I_PKI_GENERATE_RSA_KEY**,
  - **I_PKI_RSA_IN_KEY_TABLE_ONE_TAG**,
  - **I_SCRYP_AES_ENCRYPT_CBC,**
  - **I_SCRYP_AES_DECRYPT_CBC**.
- **K_PKI_RSA_PRIV_KEY** and **K_SCRYPT_AES** keys should be loaded in DEP Crypto Modules.

The configuration file containing all mandatory parameters is necessary to configure and initialize the DEP PKCS#11 library.

The DEP Cryptoki configuration file should be created manually and resided under the same directory as the DEP PKCS#11 library or DEP Cryptoki application, which uses the library.

6.1. CONFIGURATION FILE STRUCTURE

The configuration file should be an ASCII encoded text file with the use of LF[3] as line-break marker (without using carriage return CR[4] before it) and should be named "*cryptoki.cfg*".

> ***Note:***
> To convert file format with DOS(CR/LF line-break markers) line breaks to Unix line breaks(with single LF line-break markers) use free "**dos2unix"** utility.

The Cryptoki.cfg file is subdivided into the global parameters part and a number of sections. Each of these sections contains parameter-value pairs that are valid within that section.
In the latest 2.0 version of configuration file the slot sections have the highest level and contain nested DEP Platform subsections. For load balancing mechanism the 2.0 version of configuration file should be used.

The file identifies a parameter and its value in following form:

```
<global parameter name>=<global parameter value>

section {
 <slot parameter name>=<slot parameter value>

     section{
     <DEP Platform parameter name>=<DEP Platform
parameter value>
     }
}
```

> ***Note:***
> The configuration file should contain at least one slot section and each slot section should have at least one DEP Platform subsection.

All parameters are given in the following table:

Section	Name	Value
Global parameters	*version*	Configuration file version. The default value is 1.0 (in this version the configuration file does not contain any nested subsections).

[3] Line feed, '\n', 0x0A, 10 in decimal
[4] Carriage return, '\r', 0x0D, 13 in decimal

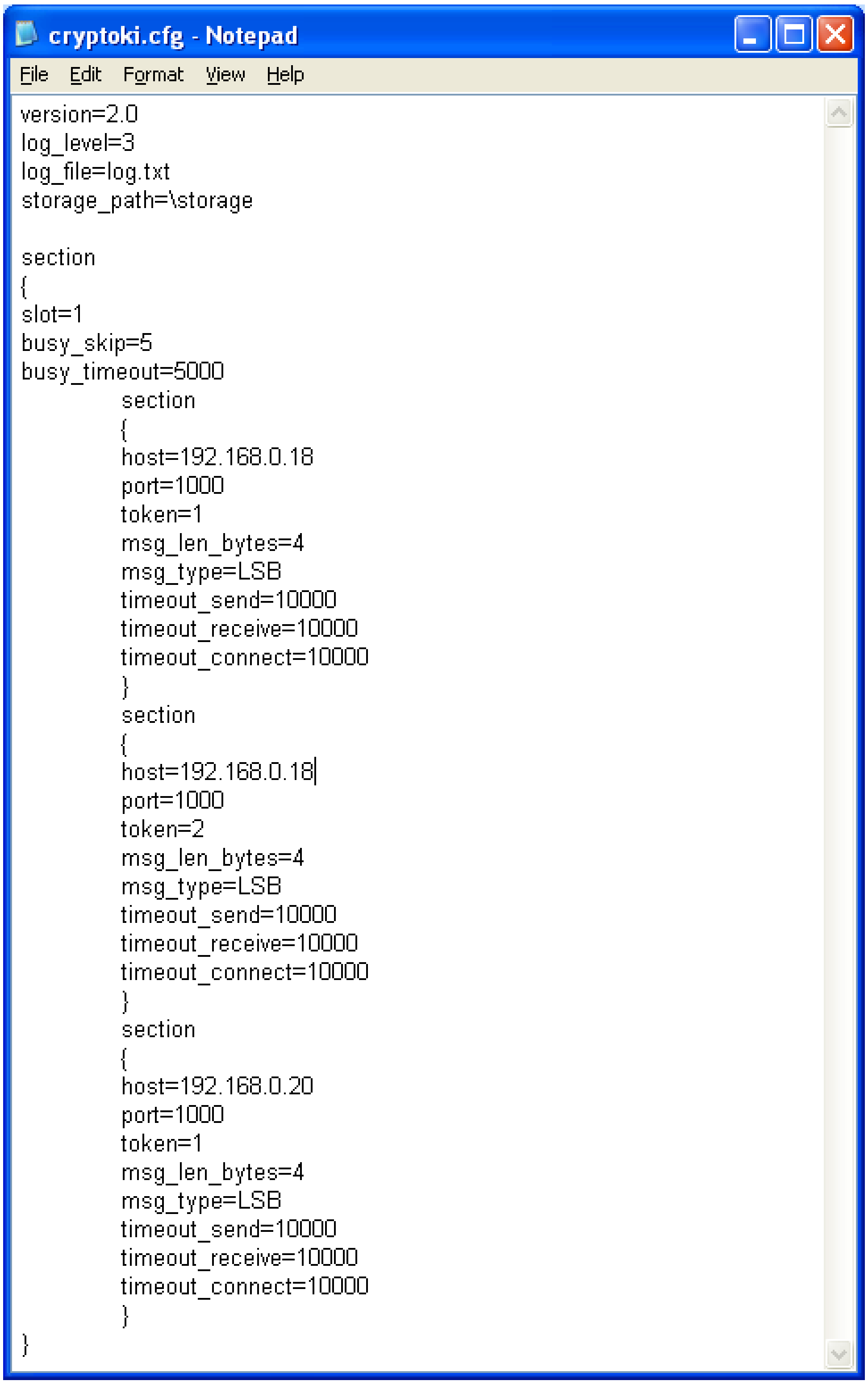
	log_level	The current logging level (possible values LOG_LEVEL_ERROR = 1 (only the error messages will be logged), LOG_LEVEL_WARNING = 2 (error and warning messages will be logged), LOG_LEVEL_INFO = 3, (error, warning and info messages will be logged), LOG_LEVEL_DEBUG = 4 (all messages will be logged)). The default value is 1.
	log_file	The file name for output log. The default file name is “log.txt”.
	storage_path	The path to store the host objects, which are supported by the DEP PKCS#11 library (certificate objects). If it is not specified, the host objects will be located in the **Storage** subfolder of installation directory in Windows and in the **depp11/storage** subfolder of **Home** folder in Linux.
Slot parameters	*slot*	The ID of the slot (starts from 0). The value is mandatory; otherwise the library initialization will be failed.
	busy_skip	The number of loops to skip busy (failed) DEP Platforms from pool in load balancing mechanism. The default value is 5.
	busy_timeout	The maximum time in milliseconds to wait to process the ECHO request to the DEP Crypto Module during the DEP Platform selection from pool in load balancing mechanism. The default busy timeout is 5000.
DEP Platform Parameters	*token*	DEP Crypto Module number (0..4) The default value is 0. In case of 0 value the transactions are processed to the DEP Platform pool.
	host	DEP Platform address. The address can be IPv4 address or the host name. The host value is mandatory: otherwise the library initialization will be failed.
	port	DEP Platform host handling port. The default value is 1000.
	msg_len_bytes	The length (in bytes) of the DEP message field (possible values 2 or 4). The default value is 4.
	msg_type	The type of the DEP message: Little or Big Endian (LSB or MSB). The default type is LSB.
	timeout_send	The timeout in milliseconds is used to send DEP message to the DEP Platform. The default value is 10000.
	timeout_receive	The timeout in milliseconds is used to receive

		DEP message from the DEP Platform. The default value is 10000.
	timeout_connect	The timeout in milliseconds is used to make connect attempt to the DEP Platform. The default value is 10000.

If any non mandatory parameter is missing in the configuration file, the default value of the parameter will be used.

Examples:

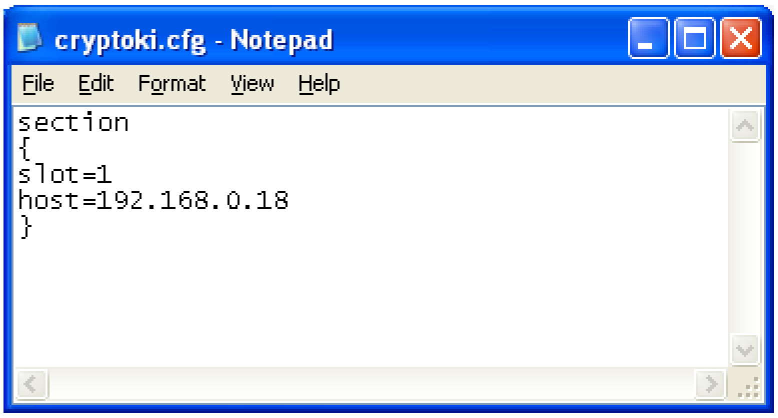
Below is an example of Cryptoki.cfg version 2.0 configuration file for one slot and three DEP Platforms using load balancing mechanism:



```
version=2.0
log_level=3
log_file=log.txt
storage_path=\storage

section
{
slot=1
busy_skip=5
busy_timeout=5000
          section
          {
          host=192.168.0.18
          port=1000
          token=1
          msg_len_bytes=4
          msg_type=LSB
          timeout_send=10000
          timeout_receive=10000
          timeout_connect=10000
          }
          section
          {
          host=192.168.0.18
          port=1000
          token=2
          msg_len_bytes=4
          msg_type=LSB
          timeout_send=10000
          timeout_receive=10000
          timeout_connect=10000
          }
          section
          {
          host=192.168.0.20
          port=1000
          token=1
          msg_len_bytes=4
          msg_type=LSB
          timeout_send=10000
          timeout_receive=10000
          timeout_connect=10000
          }
}
```

Below is an example of ready to use configuration file with minimal necessary parameters:



```
section
{
slot=1
host=192.168.0.18
}
```

7. OPENSLL/DEP PKCS#11 INTEGRATION

The DEP PKCS#11 library is integrated with OpenSSL crypto library to perform RSA sign and decrypt cryptographic operations. This integration is done by using the OpenSC project.

More information about the OpenSC project is available online at `www.opensc-project.org`.

The information and the distribution tarballs of OpenSSL released versions are available online at `www.openssl.org`.

7.1. CONFIGURING OPENSSL TO USE THE PKCS#11 DYNAMIC ENGINE

The cryptographic operations supported by the OpenSSL crypto library are being implemented either by native OpenSSL or by another dynamic engine.

The following OpenSC sub-projects are used for DEP PKCS#11 and OpenSSL integration:

- *libp11*. The libp11 is a C wrapper library for working with PKCS#11 modules;
- *engine_pkcs11;* an OpenSSL dynamic engine for PKCS#11 providers.

To be able to load the dynamic engine, the mentioned *libp11* and *pkcs11_engine* OpenSC sub-projects should be installed on the system.

The *libp11* and the *pkcs11_engine* sub-projects are stored in **openssl** subfolder under the DEP PKCS#11 installation directory.

For more information on how to install the *lib11* and the *pkcs11_engine* sub-projects refer to the http://www.opensc-project.org/libp11/wiki/QuickStart and http://www.opensc-project.org/engine_pkcs11/wiki/QuickStart web pages accordingly.
The dynamic engine can be loaded either from the OpenSSL command shell or from the OpenSSL configuration file. For more information on how to configure the OpenSSL configuration file refer to the http://www.opensc-project.org/engine_pkcs11/wiki/QuickStart web page.

To dynamically load an engine into OpenSSL execution environment, run the openssl in command shell and load the engine using the following command:

```
engine dynamic -pre SO_PATH:./engine_pkcs11.so -pre ID:pkcs11 -pre
LIST_ADD:1 -pre LOAD -pre MODULE_PATH:./libdepp11.so –pre PIN:DEP,
```

All the control commands used to load the PKCS#11 dynamic engine are given in the following table:

Parameter	**Value**
SO_PATH	The directory of *engine_pkcs11.so* file
ID	Engine ID
LIST_ADD	This command is used to add the engine to the list to be discoverable by application code later on using the engine’s ID.
LOAD	The "LOAD" command is the only one that takes no parameters and is the command that loads and uses the settings from any previous commands.
MODULE_PATH	The DEP Cryptoki library (libdepp11.so file) path
PIN	This parameter can be used to automatically authenticate a user on DEP PKCS#11 Token.

7.2. OpenSSL’s S/MIME Sign and Decrypt with DEP PKCS#11 Library

The S/MIME utility of OpenSSL crypto library is used to perform the **sign** and **decrypt** operations using the supplied certificate and the RSA private key.

7.2.1. Prerequisites

- The minimum version of DEP PKCS#11 Library must be 4.5;
- The DEP Application Software that supports the following interfaces should be loaded on DEP Crypto Module:
    - **I_STD_GET_DEP_INFO,**
    - **I_STD_GET_TAG_STATUS**,
    - **I_PKI_OUTPUT_RSA_PUBLIC_FROM_KEY_TABLE,**
    - **I_STD_ECHO,**

  - **I_PKI_RSA_MODEXP_SK**;
- The **K_PKI_RSA_PRIV_KEY** key should be loaded on DEP Crypto Module;
- The PKCS#7 formatted certificate file corresponding to the **K_PKI_RSA_PRIV_KEY** key should be present;
- An OpenSSL PKCS#11 dynamic engine should be loaded;
- An ASCII or Binary message should be given for input file when signing;
- An encrypted message in MIME format should be given for input message when decrypting.

7.2.2. S/MIME Sign with DEP PKCS#11 Library

Use the *–sign* option of *smime* command with appropriate parameters in openssl command prompt.

```
smime -sign -in data/inputmsg.txt -text -out data/signedoutputmsg.txt
-signer certificates/certificate.pem -inkey id_042507xx -keyform
engine -engine pkcs11
```

All the parameters of *sign* command are given in the following table:

Parameter	**Value**
-in *directory*/*filename*	directory and the filename of the input message to be signed
-text	adds plain text (text/plain) MIME headers to the supplied message
-out *directory*/*filename*	the directory and the filename of the output MIME format message that has been signed
-signer *directory*/*filename*	the directory and the filename of the signer’s certificate
-inkey	the ID of RSA private key object managed by the DEP PKCS#11 Library. This must match with the corresponding PKCS#7 formatted certificate
-keyform	Specifies the source of RSA private key parameter. Must take an **engine** value
-engine	loaded PKCS#11 engine’s ID

7.2.3. S/MIME Decrypt with DEP PKCS#11 Library

Use the *–decrypt* option of *smime* command with appropriate parameters in openssl command prompt.

```
smime -decrypt -in data/encryptedinputmsg.enc -out
data/decryptedoutputmsg.dec -inkey id_042507xx -keyform engine -
engine pkcs11
```

All the parameters of *decrypt* command are given in the following table:

Parameter	**Value**

-in *directory/filename*	directory and the name of the input message to be decrypted
-out *directory/filename*	the directory and the name of the message text that has been decrypted
-inkey	the ID of RSA private key object managed by the DEP PKCS#11 Library
-keyform	Specifies the source of RSA private key parameter. Must take an **engine** value
-engine	Loaded PKCS#11 engine's ID

8. LOAD BALANCING

Load balancing is a mechanism to distribute the workload evenly across two or more DEP Tokens transparently to the calling application, in order to get optimal resource utilization, minimize response time and avoid overload. If one DEP Platform or DEP Crypto Module is not available due to a failure, the operation is sent to one of the other DEP Platforms/DEP Crypto Modules. It requires additional measures to check the DEP's status and switch a failed DEP to another one (see paragraph 8.3 on page 33).

In order to take advantage of the load balancing mechanism, the Cryptoki slot maps several opened sessions to more than one DEP Platforms/DEP Crypto Modules. For example, when there are three DEP Platforms a crypto request may be sent to either DEP Platform/DEP Crypto Module based on session mapping.

During the open session when one DEP is non-responsive, one of the other resources is chosen. For this purpose, the *round-robin* method is used to choose the resource from a list of available DEP Platforms/DEP Crypto Modules. The round-robin works in a looping fashion; the DEP Platforms/DEP Crypto Modules in a pool are selected starting from the first DEP Platform/DEP Crypto Module to the last one. When the last DEP Platform/DEP Crypto Module in the list is selected it moves back to the beginning of the list. The slot should have at least one DEP Platform/DEP Crypto Module.

After the session is opened, the operations are distributed to the selected DEP Platforms/DEP Crypto Modules. If the DEP Platform/DEP Crypto Module mapped to the session fails, then all future operations on this session will fail until the DEP Platform/DEP Crypto Module is recovered, otherwise the session should be closed.

Cryptoki applications can access the DEP PKCS#11 library both in multi-threaded and mono-threaded fashion.

When the application initializes the DEP PKCS#11 library, it can specify that either it will be accessing the library from one thread, so the library need not worry about performing any type of locking for the sake of thread-safety, or it will be accessing the library concurrently from multiple threads. In this case the library must use a set of application-supplied synchronization primitives to ensure proper thread-safe behaviour. Refer to the *PKCS #11 Base Functionality v2.20* document for more information about applications and threads.

The load balancing mechanism with mono-thread and multi-thread support is described in this chapter.

8.1. LOAD BALANCING WITH MONO-THREAD SUPPORT

In mono-threaded fashion, the Cryptoki application accesses the functionalities of the DEP PKCS#11 library with one thread by using a Cryptoki slot with a pool of DEP Platforms/DEP Crypto Modules.

The Cryptoki application starts a single thread and operates with the several number of sessions opened in the context of this thread. The Cryptoki application can request a cryptographic operation on each opened session. The DEP PKCS#11 library handles these requests sequentially. When the cryptographic request is sent to the DEP, the Cryptoki application is blocked until a reply is received from the DEP Platform/DEP Crypto Module. For example, there are 2 sessions mapped to two different DEP Platforms/DEP Crypto Modules. The first session initializes a request to the DEP Platform/DEP Crypto Module and waits for reply. The Cryptoki application will be blocked and the second session cannot initialize other requests while there is an operation request waiting for reply. Only after the first session receives the reply the second session can initialize the call.
This architecture increases the reliability of DEP PKCS#11 library.

8.2. LOAD BALANCING WITH MULTI-THREAD SUPPORT

In multi-threaded fashion, the Cryptoki application accesses the functionalities of the DEP PKCS#11 library with multiple threads by using a Cryptoki slot with a pool of DEP Platforms/DEP Crypto Modules.

The DEP PKCS#11 library handles the parallelization of cryptographic functions called from different sessions distributed over many threads, which improves speed and reliability.

The Cryptoki application starts the required number of threads and operates with the sessions in the context of these opened threads. The Cryptoki application can request a cryptographic operation on each opened session. The DEP Cryptoki internally maintains the synchronization of threads that handle these requests in parallel. This architecture results in improved throughput because each session opened in different threads and mapped to the different DEP Platforms/DEP Crypto Modules can initialize function calls in parallel. For example, if there are two sessions opened in different threads, the first session initializes a request to the selected DEP Platform/DEP Crypto Module and the second session from another thread context can also initialize a request to another or to the same DEP Platform/DEP Crypto Module.

8.3. FAILOVER

The DEP PKCS#11 library supports the failover during the *open session* operation on the load balancing Cryptoki slots. If one of the DEP Platforms/DEP Crypto Modules fails in pool, the failover mechanism is used to exclude the failed DEP Platforms/DEP Crypto Modules from the pool and switch to another one.

While opening the session, a dummy request is sent to check if the DEP Platform or the DEP Crypto Module is available or not. The maximum waiting time (in milliseconds) for the response is defined in the configuration file by *busy_timeout* variable (see paragraph 6 on page 23). Failed DEP Platforms and DEP Crypto Modules will be excluded from the pool. The number of rounds to skip an unresponsive DEP Platforms/DEP Crypto Modules from the pool is defined in the configuration file by *skip_busy* parameter (see 6 on page 23).

8.4. IMPACT OF LOAD BALANCING ON KEY MANAGEMENT

To keep the transparency of the load balancing mechanism for Cryptoki applications, it is required to have the same keys loaded in all DEP Crypto Modules connected to the same Cryptoki slot.

When the session is opened on a Cryptoki slot, the operations are distributed to the pool of DEP Platforms/DEP Crypto Modules connected to it. If the key tables are not synchronized, the result of the cryptographic operation will depend on which DEP Crypto Module was addressed.

9. EJBCA/DEP PKCS#11 INTEGRATION

The DEP PKCS#11 cryptographic library can be used as a PKCS#11 provider for the Enterprise Java Beans Certificate Authority (EJBCA) OpenSource PKI. With this framework, application developers can benefit from the Java API regardless of the low-level DEP protocols.

More information on EJBCA can be found on www.ejbca.org.