PKI SDK

Developer Manual

Table of Contents

Getting Started

SDK Layout

The SDK is arranged with the following directory structure:

- **`bin`** – Compiled binaries. These include DLLs (for Windows) and other tools and utilities.
- **`doc`** – Contains this document and other reference material.
- **`include`** – Header files for using PKI from the C programming language.
- **`lib`** – Import libraries for linking with the DLLs when using MSVC.
- **`sample-code`** – Examples of how to use PKI from various programming languages. Please see the README.txt for each language for special notes on building and running the sample code. If there is no sample code for your programming language please contact Microcosm via email at **support@microcosm.com**.

Sample Code

We suggest looking at the sample code for your programming language as a starting point for getting to know the PKI SDK. Sample code is under language-specific directories in the **sample-code** directory.

Each language directory contains a README.txt explaining how to build and run the sample code.

The sample code will usually be command line based so you will need to open a Command Prompt on Windows or a Terminal on Linux/Mac in order to run the example program.

Regardless of language the sample code demonstrates two key concepts in PKI. First it attempts to create an RSA key-pair, then it goes on to sign some data using this new key-pair. In both cases the cryptography takes place in the PKI token and the RSA private key never leaves the token.

APIs

The primary API we support for communicating with PKI tokens is called **PKCS#11**. This API, which is also known as “Cryptoki”, is a standard and widely supported API originally created by RSA Security. PKCS#11 is supported on Windows through the DLLs provided in the **bin** directory of the SDK, and on Mac OSX and Linux using OpenSC.

Another API, called MS CAPI, is supported but is Microsoft specific and is Windows-only.

PKCS#11

PKCS#11 is a comprehensive and widely supported API. It can be used from most programming languages. It is a direct-call API, meaning that you simply call the library (DLL, .so etc) directly, nothing needs to be installed.

API Reference

Please see the **PKCS11_v220.pdf** in the **doc** directory of the SDK. This document is the official PKCS#11 document and it contains a detailed overview of the API.

Please note, the following functions are unavailable:

- `C_GetOperationState`
- `C_SetOperationState`
- `C_CopyObject`
- `C_GetObjectSize`
- `C_DigestKey`
- `C_DigestEncryptUpdate`
- `C_DecryptDigestUpdate`
- `C_SignEncryptUpdate`
- `C_DecryptVerifyUpdate`
- `C_DeriveKey`
- `C_SeedRandom`
- `C_GetFunctionStatus`
- `C_CancelFunction`

Calling any of these functions will result in `CKR_FUNCTION_NOT_SUPPORTED` being returned.

Supported Algorithms

Algorithm	Encryption/ Decryption	Signature Check	Hash	Key-Pair Generation	Wrap
CKM_RSA_PKCS_KEY_PAIR_GEN				✓	
CKM_RSA_PKCS	✓	✓			✓
CKM_MD2_RSA_PKCS		✓			
CKM_MD5_RSA_PKCS		✓			
CKM_SHA1_RSA_PKCS		✓			
CKM_SHA224_RSA_PKCS		✓			
CKM_SHA256_RSA_PKCS		✓			
CKM_SHA384_RSA_PKCS		✓			
CKM_SHA512_RSA_PKCS		✓			
CKM_RSA_X9_31_KEY_PAIR_GEN				✓	
CKM_RSA_X9_31		✓			
CKM_SHA1_RSA_X9_31		✓			
CKM_RSA_PKCS_OAEP	✓				
CKM_RSA_PKCS_PSS		✓			
CKM_SHA1_RSA_PKCS_PSS		✓			
CKM_SHA256_RSA_PKCS_PSS		✓			
CKM_SHA224_RSA_PKCS_PSS		✓			
CKM_SHA384_RSA_PKCS_PSS		✓			
CKM_SHA512_RSA_PKCS_PSS		✓			
CKM_RSA_X_509	✓	✓			✓
CKM_EC_KEY_PAIR_GEN				✓	
CKM_ECDSA		✓			
CKM_ECDSA_SHA1		✓			
CKM_DH_PKCS_KEY_PAIR_GEN				✓	
CKM_RC2_KEY_GEN				✓	
CKM_RC2_ECB	✓				
CKM_RC2_CBC	✓				
CKM_RC2_CBC_PAD	✓				
CKM_RC4_KEY_GEN				✓	
CKM_RC4	✓				
CKM_DES_KEY_GEN				✓	
CKM_DES_ECB	✓				
CKM_DES_CBC	✓				

Algorithm	Encryption/ Decryption	Signature Check	Hash	Key-Pair Generation	Wrap
CKM_DES_CBC_PAD	✓				
CKM_DES3_KEY_GEN				✓	
CKM_DES3_ECB	✓				
CKM_DES3_CBC	✓				
CKM_DES3_CBC_PAD	✓				
CKM_AES_KEY_GEN				✓	
CKM_AES_ECB	✓				
CKM_AES_CBC	✓				
CKM_AES_CBC_PAD	✓				
CKM_MD2			✓		
CKM_MD5			✓		
CKM_SHA_1			✓		
CKM_SHA224			✓		
CKM_SHA256			✓		
CKM_SHA384			✓		
CKM_SHA512			✓		

Supported Key Lengths

Algorithm	Key Length(s)
CKM_RSA_KEY_PAIR_GEN	1024 bits 2048 bits
CKM_RC2_KEY_GEN	1 – 128 bytes
CKM_RC4_KEY_GEN	1 – 256 bytes
CKM_DES_KEY_GEN	8 bytes
CKM_DES3_KEY_GEN	24 bytes
CKM_GENERIC_SECRET_KEY_GEN	1 – 256 bytes
CKM_AES_KEY_GEN	128 bits 192 bits 256 bits

Important Note: The tables above document all supported algorithms for completeness. Certain algorithms are no longer considered secure depending on the context in which they are used. Please review your security requirements carefully.

Language Specific Details

C/C++

Windows: Call one of the mpki DLLs in the **bin** directory of the SDK depending on whether your program is 32-bit or 64-bit. Using MSVC you can link using the import libraries in the **lib** directory of the SDK. This is what the sample code does.

Linux & Mac OSX: We recommend using OpenSC on Mac/Linux. OpenSC is an open source project that includes support for our PKI tokens. OpenSC is freely available on GitHub here: https://github.com/OpenSC/OpenSC

For details on the PKCS#11 C API please see the API Reference section above.

Java

Use the Java security APIs (ie, the classes in the `java.security` package) in conjunction with the Sun PKCS#11 provider. The Sun PKCS#11 provider uses the native PKCS#11 libraries to communicate with the PKI token. Please see the sample code for further explanation.

Note: You may wish to consider using the BouncyCastle crypto library which contains many useful APIs to aid working with PKI in Java. The sample code demonstrates using this library to create a certificate during the key-generation process.

Microsoft Cryptography API

Microsoft Cryptography API (MS CAPI) is a Windows-only alternative API that we support for using PKI tokens. CAPI employs Cryptographic Service Providers (CSPs) which perform the actual communication with crypto hardware from a particular vendor. CSPs are implemented as DLLs and must be installed in Windows before they can be called upon. The **mpki32.dll** and **mpki64.dll** in the **bin** directory of the SDK implement the CSP for our PKI tokens and can be installed by following the steps below.

CSP Module

The CSP module enables the PKI token to be used via MS CAPI.

Details of the CSP module are as follows:

- **Name**: “EnterSafe ePass2003 CSP v1.0”
- **Type**: `PROV_RSA_FULL`
  This general purpose type of CSP provides support for digital signatures and data encryption/decryption. All public key operations are performed using RSA.

API Reference

The official reference documentation for CAPI can be found on MSDN here:
http://msdn2.microsoft.com/en-us/library/aa380256.aspx

Note: The following functions are not available with our CSP:

- `CPDuplicateKey`
- `CPDuplicateHash`
- `CPHashSessionKey`

Supported Algorithms

Algorithm (`ALG_ID`)	Min Length (bits)	Max Length (bits)	Default Length (bits)	Purpose
`CALG_RC2`	8	1024	40	Encryption & Decryption
`CALG_RC4`	8	2048	40	
`CALG_DES`	56	56	56	
`CALG_3DES`	192	192	192	
`CALG_SHA1`	160	160	160	Hashing
`CALG_MD2`	128	128	128	
`CALG_MD5`	128	128	128	

Algorithm (ALG_ID)	Min Length (bits)	Max Length (bits)	Default Length (bits)	Purpose
CALG_SSL3_SHAMD5	288	288	288	
CALG_RSA_SIGN (AT_SIGNATURE)	1024	2048	1024	Signature Verification
CALG_RSA_KEYX (AT_KEYEXCHANGE)	1024	2048	1024	Encryption, decryption & signature verification

OpenSSL Integration

Overview

Microcosm PKI tokens can be integrated with OpenSSL through PKCS#11 for the purposes of digital signing, encryption and CSR creation. This is done through a PKCS#11 engine for OpenSSL which acts as a bridge between OpenSSL and the PKCS#11 module.

This chapter contains examples showing how to use the PKI token with OpenSSL in various senarios.

Prerequisites

Windows

The **bin** directory of the SDK contains 32-bit and 64-bit engine DLLs for OpenSSL on Windows. They are called **engine_pkcs11.dll** and **engine_pkcs11_64.dll** respectively.

OpenSSL for Windows can be download from
https://slproweb.com/products/Win32OpenSSL.html

The examples in this chapter assume the following:

- You unzipped the SDK to C:\PKI-SDK
- You installed the 32-bit OpenSSL Windows binaries to C:\OpenSSL-Win32

Linux

You will need the following packages installed. The Linux examples in this chapter have been tested on Debian 9 (Stretch) with the package versions indicated.

- opensc (0.16.0)
- opensc-pkcs11 (0.16.0)
- libengine-pkcs11-openssl1.1

Ensure the above packages are installed using your package manager, for example:

```
apt-get install opensc opensc-pkcs11 libengine-pkcs11-openssl1.1
```

Next, you will need to initialize the PKI token using PKCS#15 as follows:

```
pkcs15-init -E
pkcs15-init --create-pkcs15 --profile pkcs15+onepin --label "Something"
```

CSR Generation (Windows)

The following example demonstrates how to use the PKI token with OpenSSL to generate a CSR.

Firstly, open a Windows Command Prompt

At the command-prompt, change to the SDK directory using the **cd** command:

```
cd C:\PKI-SDK
```

Generate a 2048-bit RSA key-pair on the PKI token using the keygen utility:

```
bin\keygen.bat 12345678 0xAABBCC 2048
```

Next, run OpenSSL:

```
C:\OpenSSL-Win32\bin\openssl.exe
```

OpenSSL will now be running and the prompt will have changed to **OpenSSL>**.

Load the PKCS#11 engine:

```
engine dynamic -pre SO_PATH:bin\engine_pkcs11.dll -pre ID:pkcs11 -pre LIST_ADD:1 -pre
LOAD -pre MODULE_PATH:bin\mpki32.dll
```

If the command completes successfully you should see something like this at the terminal:

```
(dynamic) Dynamic engine loading support
[Success]: SO_PATH:bin\engine_pkcs11.dll
[Success]: ID:pkcs11
[Success]: LIST_ADD:1
[Success]: LOAD
[Success]: MODULE_PATH:bin\mpki32.dll
Loaded: (pkcs11) pkcs11 engine
```

Next, create a CSR using the PKI token to sign the CSR using the private key we generated previously:

```
req -new -keyform engine -engine pkcs11 -key slot_1-id_AABBCC -out demo.csr -subj
/C=GB/ST=Bristol/L=Bristol/O=Microcosm/OU=PKIDept/CN=Example/emailAddress=example@exa
mple.com
```

You should now have a CSR file called **demo.csr** in your SDK directory.

You can now quit OpenSSL:

```
exit
```

Inspect your new CSR using OpenSSL:

```
C:\OpenSSL-Win32\bin\openssl.exe req -text -noout -verify -in demo.csr
```

You should see something like this:

```
verify OK
```

```
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = GB, ST = Bristol, L = Bristol, O = Microcosm, OU = PKIDept,
 CN = Example, emailAddress = example@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c1:25:3d:76:47:7b:9a:28:f7:e3:5f:de:bf:55:
                    1d:39:4d:b0:d3:a4:15:0c:aa:b6:e6:e0:ec:99:58:
                    ae:c8:f7:20:89:a7:18:c7:c0:b5:60:bf:89:15:90:
                    03:2b:70:95:2d:87:c1:14:a7:8c:1b:6a:93:51:5f:
                    e1:74:de:f9:ec:7b:04:cc:bd:e5:57:69:ee:08:f5:
                    c3:73:95:2d:7a:bb:e3:d4:ce:71:cb:ee:02:ef:7e:
                    3c:b2:13:1a:6a:6e:ff:ad:ec:c4:69:9b:74:cf:2c:
                    10:74:a2:1e:b6:fe:ad:c0:bf:b1:0f:e9:99:bc:fe:
                    df:fc:97:b1:2a:37:c0:cc:71:30:56:e9:3b:de:c5:
                    63:5c:98:f2:9b:1f:45:f5:a4:71:b1:ab:e1:5f:51:
                    94:cd:50:df:45:a1:d8:59:a5:05:27:81:60:b5:3b:
                    fb:1d:e6:01:7a:54:15:89:88:9d:14:9d:4d:19:d5:
                    b3:7a:db:b8:2c:fb:ff:f1:e5:90:a1:07:59:f2:19:
                    2c:fe:e9:b1:ca:17:a2:9f:ea:18:dc:f9:02:cc:ac:
                    55:de:60:ca:fa:00:d1:29:a4:53:56:94:59:9b:2d:
                    13:4c:8c:79:aa:de:bf:94:ea:b9:55:b4:53:86:b8:
                    14:59:ce:1f:73:21:fd:53:da:18:5c:b5:40:4b:a4:
                    97:09
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         a7:47:4f:cf:e5:49:5c:b3:d4:f9:8f:62:35:2a:ca:b0:a0:ca:
         85:60:3c:fa:74:45:a9:76:c9:9a:e0:bd:23:d1:f5:df:92:a0:
         de:d1:fe:63:e3:54:8c:e9:0e:b2:ce:41:57:75:71:aa:88:10:
         e2:ce:06:6a:38:2f:90:0e:0d:0e:6d:e0:31:fe:2d:fd:c5:07:
         23:07:02:13:8f:85:b4:8b:18:18:3e:63:20:49:69:9f:ce:3b:
         0f:a7:d8:b7:9f:19:94:d8:97:c0:1f:c3:8b:d8:a2:7f:96:48:
```

```
        8c:50:f8:d0:9b:5b:8c:0c:3b:d2:0e:9a:f7:bc:d1:13:d0:5e:
        e1:86:c6:af:a1:9b:ae:eb:cb:17:75:fd:bb:c9:f0:10:9f:47:
        49:dd:f2:47:45:d9:ae:a1:d6:3c:71:90:04:a3:af:b9:c4:ef:
        33:7d:c5:7c:52:61:bc:1c:25:fc:c8:18:c6:b1:82:42:75:04:
        f2:77:a5:5c:01:73:1d:a3:06:30:00:6d:23:8a:1b:85:ad:4e:
        d1:68:0a:6e:b5:2d:81:59:9b:9e:51:38:22:58:5c:81:14:00:
        00:b6:25:94:3e:4e:31:89:d8:d9:b6:13:8b:6e:20:05:f8:66:
        2a:d9:4e:a4:7c:68:03:48:f9:85:c7:7d:90:e5:e4:29:e7:d0:
        2c:08:af:9a
```

This means the CSR creation was successful.

CSR Generation (Linux)

The following example demonstrates how to use the PKI token with OpenSSL to generate a CSR.

This example, assumes you have an existing PEM private key that you want to store on the token initially, before using it to generate the CSR.

First, store your PEM-encoded private key on the PKI token:

```
pkcs15-init --store-private-key privatekey.pem --id AABBCC --auth-id 01 --key-usage
sign,decrypt
```

Next, you can inspect the token to check the private key is present like this:

```
pkcs11-tool –module opensc-pkcs11.so -O
```

Which should give output like this:

```
Using slot 0 with a present token (0x0)
Public Key Object; RSA 2048 bits
  label:      Public Key
  ID:         aabbcc
  Usage:      encrypt, verify, wrap
```

Next run openssl:

```
openssl
```

and you should see the **OpenSSL>** prompt.

Next, load the PKCS#11 engine as follows:

```
engine dynamic -pre SO_PATH:/usr/lib/x86_64-linux-gnu/engines-1.1/libpkcs11.so -pre
ID:pkcs11 -pre LIST_ADD:1 -pre LOAD -pre MODULE_PATH:/usr/lib/x86_64-linux-
gnu/pkcs11/onepin-opensc-pkcs11.so
```

The correct paths to use in that command can be obtained by looking up the file locations in each package as follows:

```
dpkg -L opensc-pkcs11
dpkg -L libengine-pkcs11-openssl1.1
```

Next, you can generate a CSR using the private key on the token to sign it:

```
req -new -keyform engine -engine pkcs11 -key slot_0-id_AABBCC -out demo.csr -subj
/C=??/ST=???/L=???/O=???/CN=???/emailAddress=???
```

The format of the parameter to **-key** is slot_N-id_HEXID
where

N	is the slot number indicated in the output of **pkcs11-tool**
HEXID	is the id you passed to **pkcs15-init** when storing the private key

Alternatively you can add those two lines to a file, for example openssl.script, then run the openssl command like this:

```
openssl < openssl.script
```

Either way, you should end up with a file **demo.csr** on your disk which you can validate like this:

```
openssl req -text -noout -verify -in demo.csr
```

Data Signing (Windows)

At the command-prompt, change to the SDK directory using the **cd** command:

```
cd C:\PKI-SDK
```

Generate a 2048-bit RSA key-pair on the PKI token using the keygen utility:

```
bin\keygen.exe 12345678 0xAABBCC 2048
```

If you have already created the demo key-pair then the keygen command will tell you:

```
Token already contains a key with that ID
```

This is fine, so carry on.

Next, run OpenSSL:

```
C:\OpenSSL-Win32\bin\openssl.exe
```

OpenSSL will now be running and the prompt will have changed to **OpenSSL**>.

Load the PKCS#11 engine:

```
engine dynamic -pre SO_PATH:bin\engine_pkcs11.dll -pre ID:pkcs11 -pre LIST_ADD:1 -pre
LOAD -pre MODULE_PATH:bin\mpki32.dll
```

If the command completes successfully you should see this at the terminal:

```
(dynamic) Dynamic engine loading support
[Success]: SO_PATH:bin\engine_pkcs11.dll
[Success]: ID:pkcs11
[Success]: LIST_ADD:1
[Success]: LOAD
[Success]: MODULE_PATH:bin\mpki32.dll
Loaded: (pkcs11) pkcs11 engine
```

Next, call the digest command in OpenSSL to calculate the digital signature of a file (README.txt in this example), using the private key on the token:

```
OpenSSL> dgst -engine pkcs11 -sign slot_1-id_AABBCC -keyform engine -hex -sha256
README.txt
```

If the command is successful you should see the output of the signature at your terminal (your actual signature will be different):

```
RSA-SHA256(README.txt)= bf151d4a3d5a67800fb1e581931baee4c884d1d8ff4848542261b41
34d1c7e596d21813f6e2f161ab01c86373ca2731f1d3325f1a3ff400ebec1d36faeda7da71e71e2
9d60533c94e2c1bb528d9522cf2bbd509e2e42e089e267a6c3bcb453d982454f2d1ce10ff3af883
a5abdbca46af07e791eb05c1e78ab14943b3258aeebaad92eb4ab3a562e078c8fea53873aca5be0
6a20ef4ee57e8cf1a7a79f3c8246de7960793dda7584c71f8205e92da017fc41d1c61aba84f121d
```

```
632c9a05ab238089fac225ac83544480b1f212a3729f3527787695030f006ad98e6d3fc973bc2c4
8006e972e6583068e708c8eae3052e1244f69c4cbe6186e2afe767ec
```

LUKS Integration

The PKI token can be used to protect the encryption key used to encrypt disks and partitions using LUKS (via cryptsetup).

For documentation of how to set this up, please see the **LinuxDiskEncryption.pdf** in the **docs** directory of the SDK.

Utilities

The following tools are located in the **bin** directory of the SDK.

Key Generation (keygen)

This tool allows you to create named (aliased) keys in the PKI token. The tool is called **keygen.bat**.

Usage is as follows:

```
      keygen -i ID -n KEYSIZE [-p PIN]
where...
      ID is the ID/ALIAS of the new key
      KEYSIZE is the required size (in bits) of the RSA key
      PIN is the PIN of the PKI token (keygen will prompt for pin by default)
```

Certificate Import (import-cert)

This tool allows you to import a public-key certificate into the token. The tool is called **import-cert.bat**.

Usage is as follows

```
      import-cert FILENAME
where...
      FILENAME is the file name of the certificate to import
```

PKI Token Explorer (pkcs11-explorer)

The tool lets you inspect and manage the token.

Usage:

```
      pkcs11-explorer
```

After logging in to the token using the User PIN, you will be given some options:

```
Options...
        (0) Quit
        (1) List mechanisms supported by the token
        (2) List objects in token
        (3) Delete object from token
        (4) Delete all objects from token
```

Support

If you have any questions about the PKI product please contact Microcosm using one of following methods.

Technical Support & General Enquiries

Email: support@microcosm.com

Telephone: +44 (0) 117 983 0084

Sales/Ordering

Email: sales@microcosm.com

Telephone: +44 (0) 117 983 0084