Entegrity PKCS#11 Workbench

Description

Version 1.4 – 9 Oct 2000

Overview

The Entegrity PKCS#11 Workbench is a C/C++ program. Source is provided. The workbench has been used in both Wintel and Solaris environments.

The program is executed from the command line with a single parameter, the pathname of the appropriate PKCS#11 DLL/SO.

The program outputs a log as it performs the tests, with detailed error reports should an error be detected.

Test Descriptions

The functionality of the workbench is split into 4 sets of tests:

- Basic Tests – these test the data object handling, sessions and PIN functionality
- Cryptographic Tests – these test key generation and asymmetric and symmetric cryptographic functions
- Status and Information Tests
- Advanced Test – this tests the C_WaitForSlotEvent function

Basic tests

The following table summarizes the basic tests that are performed.

Test Step	PKCS#11 functions called
Initialise the token using 1111 as SO password and 1111 as user password	C_InitToken (set SO PIN = 1111) C_OpenSession C_login (as SO PIN =1111) C_InitPin (set user PIN=1111) C_logout (of SO session) C_CloseSession
Open a Session;	C_OpenSession (as a public session)
Create small Public Object hObj1	C_CreateObject(class = CKO_DATA, private=FALSE) Without the CKA_APPLICATION set.
Check that hObj1 and values exist on Token	C_GetAttributeValue should give an empty string for CKA_APPLICATION.
Destroy the public data object hObj1	C_DestroyObject
Create small Public Object hObj1	C_CreateObject(class = CKO_DATA, private=FALSE)

Unsuccessful attempt to create a small private data object during a public session	C_CreateObject(class = CKO_DATA, private=TRUE)
Unsuccessful attempt to login as a regular user with wrong password	C_login (as User PIN =blah)
Login as normal user with password 1111	C_login (as User PIN =1111)
Create a large private data object - hObj2	C_CreateObject(class = CKO_DATA, private=TRUE)
Check that hObj1 and values exist on Token	C_GetAttributeValue
Check that hObj2 and values exist on Token	C_GetAttributeValue
search for private data objects should return hObj2	C_FindObjectsInit (for private objects) C_FindObjects C_FindObjectsFinal
logout the normal user session	C_Logout
check can access the public data object hObj1	C_FindObjectsInit C_FindObjects C_FindObjectsFinal
Unsuccessful attempt to checking for private data object hObj2 (as not logged on);	C_FindObjectsInit C_FindObjects C_FindObjectsFinal
Destroy the public data object hObj1	C_DestroyObject
Unsuccessful attempt to destroy the public data object hObj1 again	C_DestroyObject
A search for public data objects should return none – hObj1 having been deleted	C_FindObjectsInit C_FindObjects C_FindObjectsFinal
Unsuccessful attempt to destroy the private data object hObj2 (as not logged in as a user)	C_DestroyObject
login as regular user with password 1111	C_login (as User PIN =1111)
fetch the token info	C_GetTokenInfo
Unsuccessful try to change password with the wrong old password	C_SetPin(old = Blah, new = something)
IF SUPPORTS MIN SIZE PASSWORD THEN unsuccessful try to change password with too small new password	C_SetPin
IF SUPPORTS MIN SIZE PASSWORD THEN Unsuccessful try to change password with too big new password	C_SetPin
Finally, change the password to new value	C_SetPin(old = 1111, new = something)
Search for private data objects should still return hObj2	C_FindObjectsInit C_FindObjects C_FindObjectsFinal
Logout	C_logout

Unsuccessful try to login with the wrong password (in fact the previous password)	C_Login(as User PIN = 1111)
Login with the correct password	C_Login(as User PIN = something)
A search for private data objects should still return hObj2	C_FindObjectsInit C_FindObjects C_FindObjectsFinal
Modify the private data object hObj2	C_SetAttributeValue
Check the modifications were applied	C_GetAttributeValue
Destroy the private data object hObj2	C_DestroyObject
Change the password to the initial one allocated 1111	C_Login(as User PIN = 1111)
Logout	C_Logout
Close session	C_CloseSession

Cryptographic tests

In these series of tests the workbench tries to generate two RSA and triple DES keys, if the keys are supported.

Test Step	PKCS#11 functions called
Open a R/W session s1	C_OpenSession, session for key generation.
Login	C_Login, PIN=1111
Remove all objects	C_DestroyObject, first destroy all public token objects, then all private token objects
Open a R/O session s2	C_OpenSession, sign/verify/encrypt/decrypt are performed on a separate R/O session.
Generate two RSA key pair of 1024 size	C_GetMechanismInfo to see if the 1024 RSA keygen is supported C_GenerateKeyPair with the following templates: Private key: ❑ CKA_TOKEN, true ❑ CKA_PRIVATE, true ❑ CKA_SENSITIVE, true ❑ CKA_EXTRACTABLE, false ❑ CKA_MODIFIABLE, true ❑ CKA_DECRYPT, true ❑ CKA_SIGN, true Public key: ❑ CKA_PUBLIC_EXPONENT, 0x010001 ❑ CKA_ENCRYPT, true

	❑ CKA_VERIFY, true ❑ CKA_MODULUS_BITS, 1024
Confirm the generated key	C_FindObjects, verify that the generated private key has the desired ID
Generate the pseudo-random data	C_GenerateRandom, random data for sign/verify.
Sign the data	C_GetMechanismInfo, check whether the mechanism is supported is CKF_SIGN_RECOVER. C_SignRecoverInit, initialize the signature operation with data recovery. C_SignRecover, signs the data in a single operation. C_GetMechanismInfo, check whether the mechanism is supported is CKF_SIGN C_SignInit, , initialize the signature operation and the key supports signatures with appendix. C_Sign, signs the data in a single part.
Verification of the data	C_GetMechanismInfo, check whether the mechanism is supported is CKF_VERIFY_RECOVER C_VerifyRecoverInit, initialize the signature verification operation with data recovery. C_VerifyRecover, verify a signature in a single operation. C_GetMechanismInfo, check whether the mechanism supported is CKF_VERIFY C_ VerifyInit, , initialize the signature verification operation and the signature is an appendix to the data. C_Verify, verifies the signature in a single part.
Generate the pseudo-random data	C_GenerateRandom, random data for encrypt/decrypt.
Encryption	C_GetMechanismInfo, check whether the mechanism supported is CKF_ENCRYPT C_Encrypt, encrypts the data.
Decryption	C_GetMechanismInfo, check whether the mechanism supported is CKF_DECRYPT C_Decrypt, decrypts the data.
Confirm the generated key	C_FindObjects, make sure that all generated private keys are still on the token.
Remove all objects	C_DestroyObject , destroy all public token objects, then all private token objects.
Generate two triple DES key of 64 bit size	This keys are not store on the token. C_GetMechanismInfo to see if the 64 bit DES keygen is supported C_GenerateRandom, create a keyid from a random number. C_GenerateKey with the following templates:

	Symmetric key: ❑ CKA_TOKEN, false ❑ CKA_SENSITIVE, true ❑ CKA_ENCRYPT, true ❑ CKA_DECRYPT, true ❑ CKA_SIGN, true ❑ CKA_VERIFY, true ❑ CKA_WRAP, true ❑ CKA_UNWRAP, true ❑ CKA_ID, keyid
Confirm the generated key	C_FindObjects, verify that the generated secret key has the desired ID
Generate the pseudo-random data	C_GenerateRandom, random data for sign/verify.
Sign the data	C_GetMechanismInfo, check whether the mechanism is supported is CKF_SIGN_RECOVER. C_SignRecoverInit, initialize the signature operation with data recovery. C_SignRecover, signs the data in a single operation. C_GetMechanismInfo, check whether the mechanism is supported is CKF_SIGN C_SignInit, , initialize the signature operation and the key supports signatures with appendix. C_Sign, signs the data in a single part.
Verification of the data	C_GetMechanismInfo, check whether the mechanism is supported is CKF_VERIFY_RECOVER C_VerifyRecoverInit, initialize the signature verification operation with data recovery. C_VerifyRecover, verify a signature in a single operation. C_GetMechanismInfo, check whether the mechanism supported is CKF_VERIFY C_ VerifyInit, , initialize the signature verification operation and the signature is an appendix to the data. C_Verify, verifies the signature in a single part.
Generate the pseudo-random data	C_GenerateRandom, random data for encrypt/decrypt.
Encryption	C_GetMechanismInfo, check whether the mechanism supported is CKF_ENCRYPT C_Encrypt, encrypts the data.
Decryption	C_GetMechanismInfo, check whether the mechanism supported is CKF_DECRYPT C_Decrypt, decrypts the data.
Confirm the generated key	C_FindObjects, make sure that all generated secret keys

	are still exists in the session.
Close session	C_CloseSession, close the session s2.
Logout	C_Logout, logout from session s1.
Close session	C_CloseSession, close the session s1.

Status and Information tests

The workbench also lists the token information (version information and the manufacturer) and the list of mechanisms supported.

Test Step	PKCS#11 functions called
List the Cryptoki interface version number, library description and the manufacture.	C_GetInfo, general information about Cryptoki.
List the token flags, pin length, total memory and the free memory.	C_GetTokenInfo, information about a particular token.
List the supported mechanisms	C_GetMechanismList, list of mechanisms supported by a token.

Advanced Test

Finally there is an advanced test to test the C_WaitForSlotEvent which test the asynchronous notification of token status, and is optional.

Test Step	PKCS#11 functions called
Test the asynchronous notification of token status.	C_WaitForSlotEvent