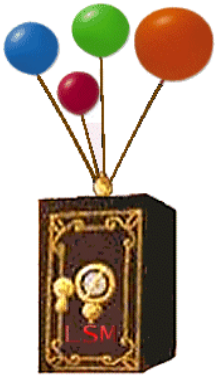


LSM-PKCS#11

Lite Security Module

A daemon to handle Secure Boxes (cryptographic keys, X509 certificates and data objects), accessible through a PKCS#11 library, supporting non-certified (lite) Hardware or Software Security Modules.

Version 1.0.1

by Clizio Merli
(clizio@clizio.com)

To my wife Laura and my son Simon.

Table of Contents

1. General Overview

LSM-PKCS11 is a package intended to support the implementation of **Lite Security Modules**, i.e. a kind of *not certified* [1] Software or Hardware Security Modules (HSM, SSM). The targets of such implementations are PKIs (Public Keys Infrastructures) for intra-company and network applications, requiring a non-trivial security level but not so 'budgeted' to allow the acquisition of true (certified) HSMs, whose cost starts from as little as some thousands dollars.

The basic component of LSM-PKCS11 is a **multi-threaded daemon** that can be hosted on a little dedicated system, running Linux (or if you like it, Windows NT/2000XP as well), to support a set of cryptographics operations released by OpenSSL library (www.openssl.org) on some well-protected files (Security Boxes) hosting cryptographics items like public and private keys, secret keys, data objects, certificates and so on.

The daemon services can be accessed via a TCP/IP connection with the support of a **shared library** (DLL in Windows environment) conforming to the **PKCS#11 standard** developed by RSA Laboratories (http://www.rsasecurity.com), also known as Criptoki[2]. PKCS#11 is part of the Public-Key Cryptography Standards (PKCS).

Developing LSM-PKCS#11 I tried to adhere as much as possible to the PKCS#11 standard specifications, so to allow a full integration with applications using PKCS#11 interface to access security tokens for digital signature, verification, and other cryptographic facilities.

The first version of the package supports only the minimum of cryptographic mechanisms:

RSA, DSA, DES and DES3 encryption and decryption,
RSA and DSA digital signature and verification,
MD2, MD5 and SHA1 digesting,
random generation.

But after full initial testing, the package will be easily extended to support more and more cryptographic mechanisms.

The initial deployement didn't cost too much (just a month of evenings and holidays, thanks to the patience of my wife Laura). But after the first step any help is welcome, in the aim to consolidate the package. Remaing activities are:

the deployment of a serious test environment,
the extension of cryptographic capabilities,
the deployment of configuration utilities.

You're welcome!

Clizio Merli

[1] HSMs (or SSMs) used to support cryptographic operations in the communication among different entities (private and public companies, associations, and so on) must be certified in adherence to certification schemas (FIPS-140, ITSEC, …) recognized by national and international authorities. Such certifications add a strong value to these devices, but at the same time raise their commercial price to some thousands of dollars. In many intra-company or network applications a secure device is a desired requirement, but the allocated budgets are not so big to allow the acquisition of a certified HSM, leaving the maintenance of cryptographic objects (signing keys) on trivial data files, with a low security level. LSM fulfils this gap, allowing the implementation of non-certified, low cost HSMs, with a relatively high security level.
[2] PKCS#11: Cryptographic Token Interface Standard.

2. LSM-PKCS11 Architecture

The architecture of LSM-PKCS11 is that of a standard client/server application, in which the client is any user process using the PKCS#11 library, and the server is the LSM daemon running somewhere in the network (for HSMs, Hardware Security Modules), or in the simplest case on the same machine as client (for SSMs, Software Security Modules).

The communication channel between the client and the server is a standard TCP connection, enforced with:

- a compression mechanism to reduce the network bandwith,
- a session layer implementing the PKCS#11 session protocol,
- a presentation layer designed to support the PKCS#11 session protocol.

The end-points of the TCP connection are two threads, one of the user process using the PKCS#11 library, and one of the LSM daemon.

Figure 1 shows such a scenario, evidencing the protocol layers in a ISO/OSI fashion.

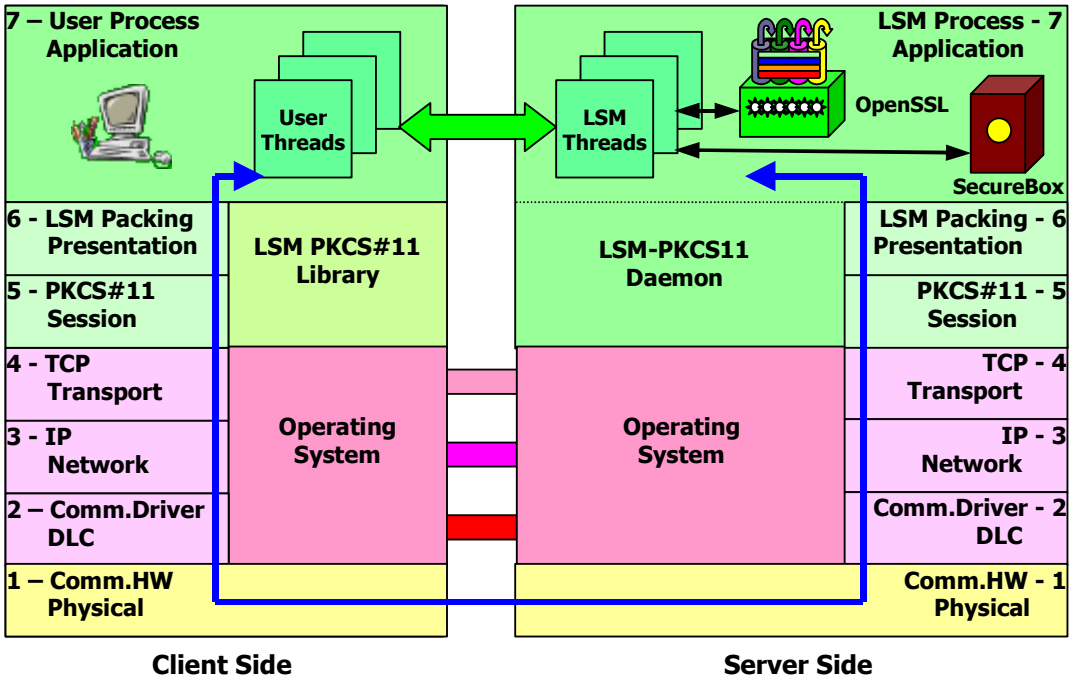


Figure 1 – LSM-PKCS11 Network Architecture

PKCS#11 requests (issued by a user process thread) are mapped, through the TCP connection, to the corresponding LSM-PKCS11 daemon thread, which performs the required functions and maps the responses back to the original user thread.

Internally the structure of LSM-PKCS11 daemon is strictly adherent to PKCS#11 standard, so to easy the interface of PKCS#11 library to the actual cryptographic information and operations. In this sense the network mappings of interface data transported over the TCP connections are a sort of system-independent mapping of PKCS#11 functions.

2.1 SecureBox

LSM-PKCS11 daemon keeps cryptographic items (public and private keys, secret keys, data objects, certificates) in special files called SecureBoxes. A SecureBox is:

- a formatted data file,
- specially designed to held cryptographic and descriptive information of secure user items (public and private keys, secret keys, data objects, certificates),
- with an intrinsic security level, independent of the security mechanism by which the file itself is protected, and
- platform independent (i.e., it may be generated on a system and used on any other system, no matter how the CPUs of these systems are similar or different in term of word size, byte endianess, or other peculiarities).

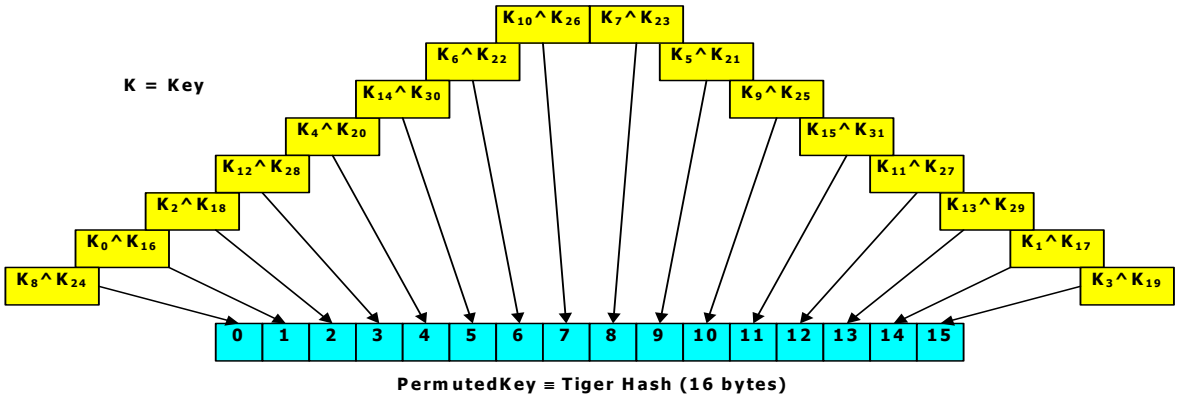
Currently the SecureBoxes handled by LSM-PKCS11 may contain up to 256 objects, for a maximum of 1048576 bytes of cryptographic and descriptive data (1 Mbyte), but these limits can be easily extended to more important figures if the final application requires a higher number of cryptographic items.

Internally a SecureBox is organized in three sections, as shown in the following table.

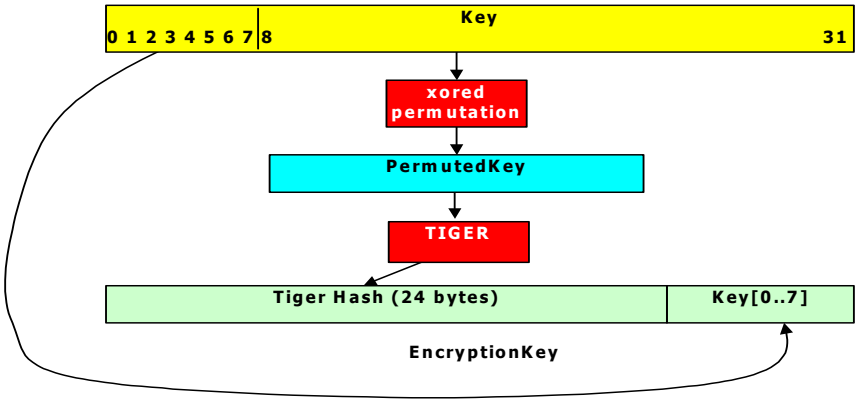
Header	general and dimensioning information		
	Magic		magic string "LSMSBOXM"
	VersionMajor		version major number
	VersionMinor		version minor number
	Name[16+1]		module name
	Key		encryption Camellia key (256 bits)
	SOPINToBeChanged		SO PIN ToBeChanged flag
	SOPINLoginCount		SO PIN erroneous login count
	SOPINMaxTrials		SO PIN max number of erroneous logins
	SOPIN		current SO PIN
	USPINToBeChanged		user PIN ToBeChanged flag
	USPINLoginCount		user PIN erroneous login count
	USPINMaxTrials		user PIN max number of erroneous logins
	USPIN		current user PIN
Object Descriptors Pool	detailed description of all objects held in the box		
	Pool	NumObjects	number of objects in module
		MaxObjects	max number of allocatable objects
		AllSize	total allocated size in data area
		FreSize	free allocatable size in data area
		MaxSize	max allocatable size in data area
	Object	Name	name of the object
		Type	object type: application dependent
		Scope	public | private | secret
		ActSize	actual object size
		AllSize	allocated size in data area
		RelPtr	relative pointer in data area
Data Area	actual cryptographic and descriptive information of held objects		
	a set of allocation units (16 bytes each): for the default value of 1 Mbyte data area (1048576 bytes), 65536 allocation units		

Every SecureBox is properly self-encrypted with a dedicated schema based on Camellia[3] algorithm and on Tiger[4] hashing mechanism. The Camellia encryption key is derived from the Key written in the header by means of a xored-permutation and a Tiger hashing:

PermutedKey = xored-permutation(Key)



EncryptionKey = Tiger(PermutatedKey) | Key[0..7]



LSM-PKCS11 daemon may handle up to four SecureBoxes: each SecureBox is presented as a separate PKCS11 slot, loaded with its token.

A set of utilities allows:

- the preparation (formatting) of a SecureBox,
- the rebuild of a SecureBox (garbaging deallocated data space),
- the dump of a SecureBox,
- the management of SO and user PINs.

[3] Camellia Block Encryption Algorithm, Version M1.02 September 24 2001, Copyright Mitsubishi Electric Corp 2000-2001. Camellia algorithm module used in LSM is derived from the original code furnished in open source by Nippon Telegraph & Telephone Corp. and Mitsubishi Electric Corp. See 'http://info.isl.ntt.co.jp/crypt/eng/camellia/technology.html'.
[4] Tiger: A Fast New Cryptographic Hash Function (Designed in 1995), Copyright Ross Anderson and Eli Biham. Tiger algorithm module used in LASM is derived from the original code furnished in open source by Ross Anderson and Eli Biham. See 'http://www.cs.technion.ac.il/~biham/Reports/Tiger/'.

2.2 Session and Object Management

Each LSM thread handles a session with end-user application threads. Session management is performed with the support of a pool of session descriptors and a dynamic queue holding currently active sessions: each active session can be accessed by one or more end-user threads (of the same user process).

A session descriptor has the layout shown in the following table.

Forward	dynamic link fro memory queue management
Backward	dynamic link fro memory queue management
Handle	logical session identifier (0..n)
SlotId	logical slot identifier (1..4)
ThreadId	LSM thread who has created the session
Info	session information (adhering PKCS#11 CK_SESSION_INFO structure)
UserType	session open mode (CKU_SO | CKU_USER)
NumObjs	number of objects currently accessed by the session
ObjectList	array of object handles currently accessed by the session
EncryptState	pointer to internal encrypting structure (if encryption active)
EncryptMechanism	encryption mechanism type (if encryption active)
DecryptState	pointer to internal decrypting structure (if decryption active)
DecryptMechanism	decryption mechanism type (if decryption active)
DigestState	pointer to internal digesting structure (if digest active)
DigestMechanism	digest mechanism type (if digest active)
SignState	pointer to internal signing structure (if signature active)
SignMechanism	signature mechanism type (if signature active)
VerifyState	pointer to internal verify structure (if verification active)
VerifyMechanism	verification mechanism type (if verification active)

Objects accessed by end-user threads are held in memory in a pool embedded in each slot memory mapping: this means that no objects can be created outside of a slot context.

An object can be (as from PKCS#11 standard) a session object (volatile, erased on session termination), or a token object (permanent, its memory content is always mapped to the corresponding SecureBox).

An object descriptor has the layout shown in the following table.

Handle	logical object identifier (1..256)
Owner	i.e. object type (session | token)
Name	logical object name
NumSessions	number of sessions currently accessing the object
AttrSet	dynamic array of session attributes
Obj	packed object content

The dynamic array of session attributes is a memory structure which helds the object contents in a form suitable for run-time memory handling of objects. This form is strictly related to the form defined by PKCS#11 standard, and easily maps to the platform independent format of object attributes when written to a SecureBox or transported over the network.

A dynamic array of session attributes has the layout shown in the following table.

NumAttrs	number of object attributes
Set	array of object attributes

where each object attribute has the layout:

Assigned	attribute value assiged by end-user (1), set to default (0)
Type	object type (boolean, numeric value, buffered value)
Size	object size in bytes: 1 if boolean, 4 if numeric value, variable if buffer
Value	actual attribute value – a union of: B boolean value V numeric value P buffered value (pointer to a dynamic memory buffer)

The type ob an object can assume one of the following values:

OBJ_T_DATA	generic user data object
OBJ_T_CERTIFICATE	X.509 certificate
OBJ_T_ATTRCERTIFICATE	X.509 attribute certificate
OBJ_T_PUBLIC_KEY_RSA	public RSA key
OBJ_T_PRIVATE_KEY_RSA	private RSA key
OBJ_T_PUBLIC_KEY_DSA	public DSA key
OBJ_T_PRIVATE_KEY_DSA	private DSA key
OBJ_T_PUBLIC_KEY_EC	public ECDSA key (future use)
OBJ_T_PRIVATE_KEY_EC	private ECDSA key (future use)
OBJ_T_SECRET_KEY_DES	secret DES key
OBJ_T_SECRET_KEY_DES3	secret DES3 key

Objects managed by LSM-PKCS11 are defined with the following attribute sets.

Clock		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE(4)	CKH_CLOCK
CKA_VALUE	CK_CHAR[16]	YYYYMMDDhhmmss00

Counter		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_HW_FEATURE
CKA_HW_FEATURE_TYPE	CK_HW_FEATURE(4)	CKH_MONOTONIC_COUNTER
CKA_RESET_ON_INIT	CK_BBOOL(1)	T|F
CKA_HAS_RESET	CK_BBOOL(1)	T|F
CKA_VALUE	Byte Array	counter big endian

Data		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_DATA
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_APPLICATION	RFC2279 string	Application
CKA_OBJECT_ID	Byte Array	DER of obj id
CKA_VALUE	Byte array	Value

X.509 Certificate		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_CERTIFICATE
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE(4)	CKC_X_509
CKA_TRUSTED	CK_BBOOL(1)	T|F
CKA_SUBJECT	Byte array	DER of subject name
CKA_ID	Byte array	Key-id of public/private key pair
CKA_ISSUER	Byte array	DER of issuer name
CKA_SERIAL_NUMBER	Byte array	DER of serial number
CKA_VALUE	Byte array	BER of certificate

X.509 AttributeCertificate		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_CERTIFICATE
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE(4)	CKC_X_509_ATTR_CERT
CKA_TRUSTED	CK_BBOOL(1)	T|F
CKA_OWNER	Byte Array	DER of subject field
CKA_AC_ISSUER	Byte Array	DER of issuer field
CKA_SERIAL_NUMBER	Byte Array	DER of serial number
CKA_ATTR_TYPES	Byte Array	BER of obj-ids values
CKA_VALUE	Byte Array	BER of certificate

RSA Public Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PUBLIC_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_RSA
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of key subject
CKA_ENCRYPT	CK_BBOOL(1)	T|F
CKA_VERIFY	CK_BBOOL(1)	T|F
CKA_VERIFY_RECOVER	CK_BBOOL(1)	T|F
CKA_WRAP	CK_BBOOL(1)	T|F
CKA_TRUSTED	CK_BBOOL(1)	T|F
CKA_MODULUS	Big integer	Modulus n
CKA_MODULUS_BITS	CK_ULONG(4)	Length in bits of modulus n
CKA_PUBLIC_EXPONENT	Big integer	Public exponent e

RSA Private Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PRIVATE_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_RSA
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of certificate subject
CKA_SENSITIVE	CK_BBOOL(1)	T|F
CKA_SECONDARY_AUTH	CK_BBOOL(1)	T|F (Deprecated)
CKA_AUTH_PIN_FLAGS	CK_FLAGS(4)	Zero (Deprecated)
CKA_DECRYPT	CK_BBOOL(1)	T|F
CKA_SIGN	CK_BBOOL(1)	T|F
CKA_SIGN_RECOVER	CK_BBOOL(1)	T|F
CKA_UNWRAP	CK_BBOOL(1)	T|F
CKA_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_ALWAYS_SENSITIVE	CK_BBOOL(1)	T|F
CKA_NEVER_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_MODULUS	Big integer	Modulus n
CKA_PUBLIC_EXPONENT	Big integer	Public exponent e
CKA_PRIVATE_EXPONENT	Big integer	Private exponent d
CKA_PRIME_1	Big integer	Prime p
CKA_PRIME_2	Big integer	Prime q
CKA_EXPONENT_1	Big integer	Private exponent d modulo p-1
CKA_EXPONENT_2	Big integer	Private exponent d modulo q-1
CKA_COEFFICIENT	Big integer	CRT coefficient q-1 mod p

DSA Public Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PUBLIC_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_DSA
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of key subject
CKA_ENCRYPT	CK_BBOOL(1)	T|F
CKA_VERIFY	CK_BBOOL(1)	T|F
CKA_VERIFY_RECOVER	CK_BBOOL(1)	T|F

CKA_WRAP	CK_BBOOL(1)	T|F
CKA_TRUSTED	CK_BBOOL(1)	T|F
CKA_PRIME	Big integer	Prime p, 512-1024 bits, step 64
CKA_SUBPRIME	Big integer	Subprime q (160 bits)
CKA_BASE	Big integer	Base g
CKA_VALUE	Big integer	Public value y

DSA Private Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PRIVATE_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_DSA
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of certificate subject
CKA_SENSITIVE	CK_BBOOL(1)	T|F
CKA_SECONDARY_AUTH	CK_BBOOL(1)	T|F (Deprecated)
CKA_AUTH_PIN_FLAGS	CK_FLAGS(4)	Zero (Deprecated)
CKA_DECRYPT	CK_BBOOL(1)	T|F
CKA_SIGN	CK_BBOOL(1)	T|F
CKA_SIGN_RECOVER	CK_BBOOL(1)	T|F
CKA_UNWRAP	CK_BBOOL(1)	T|F
CKA_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_ALWAYS_SENSITIVE	CK_BBOOL(1)	T|F
CKA_NEVER_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_PRIME	Big integer	Prime p, 512-1024 bits, step 64
CKA_SUBPRIME	Big integer	Subprime q (160 bits)
CKA_BASE	Big integer	Base g
CKA_VALUE	Big integer	Private value x

Elliptic Curves Public Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PUBLIC_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_ECDSA | CKK_EC
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of key subject
CKA_ENCRYPT	CK_BBOOL(1)	T|F

CKA_VERIFY	CK_BBOOL(1)	T|F
CKA_VERIFY_RECOVER	CK_BBOOL(1)	T|F
CKA_WRAP	CK_BBOOL(1)	T|F
CKA_TRUSTED	CK_BBOOL(1)	T|F
CKA_EC_PARAMS	Byte array	DER ANSI X9.62 param value
CKA_EC_POINT	Byte array	DER ANSI X9.62 EC point value Q

Elliptic curves DSA Private Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_PRIVATE_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_ECDSA | CKK_EC
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SUBJECT	Byte array	DER of certificate subject
CKA_SENSITIVE	CK_BBOOL(1)	T|F
CKA_SECONDARY_AUTH	CK_BBOOL(1)	T|F (Deprecated)
CKA_AUTH_PIN_FLAGS	CK_FLAGS(4)	Zero (Deprecated)
CKA_DECRYPT	CK_BBOOL(1)	T|F
CKA_SIGN	CK_BBOOL(1)	T|F
CKA_SIGN_RECOVER	CK_BBOOL(1)	T|F
CKA_UNWRAP	CK_BBOOL(1)	T|F
CKA_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_ALWAYS_SENSITIVE	CK_BBOOL(1)	T|F
CKA_NEVER_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_EC_PARAMS	Byte array	DER ANSI X9.62 param value
CKA_VALUE	Big integer	ANSI X9.62 private value d

DES Secret Key		
CKA_CLASS	CK_OBJECT_CLASS(4)	CKO_SECRET_KEY
CKA_TOKEN	CK_BBOOL(1)	T token object; F session object
CKA_PRIVATE	CK_BBOOL(1)	T private object; F public object
CKA_MODIFIABLE	CK_BBOOL(1)	T|F
CKA_LABEL	RFC2279 string	Description
CKA_KEY_TYPE	CK_KEY_TYPE(4)	CKK_DES | CKK_DES2 | CKK_DES3
CKA_ID	Byte array	Key identifier for key
CKA_START_DATE	CK_DATE(8)	Start date for the key
CKA_END_DATE	CK_DATE(8)	End date for the key
CKA_DERIVE	CK_BBOOL(1)	T|F
CKA_LOCAL	CK_BBOOL(1)	T|F
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE(4)	Id of generation mechanism
CKA_SENSITIVE	CK_BBOOL(1)	T|F
CKA_SECONDARY_AUTH	CK_BBOOL(1)	T|F (Deprecated)
CKA_AUTH_PIN_FLAGS	CK_FLAGS(4)	Zero (Deprecated)
CKA_ENCRYPT	CK_BBOOL(1)	T|F

CKA_DECRYPT	CK_BBOOL(1)	T|F
CKA_SIGN	CK_BBOOL(1)	T|F
CKA_VERIFY	CK_BBOOL(1)	T|F
CKA_WRAP	CK_BBOOL(1)	T|F
CKA_UNWRAP	CK_BBOOL(1)	T|F
CKA_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_ALWAYS_SENSITIVE	CK_BBOOL(1)	T|F
CKA_NEVER_EXTRACTABLE	CK_BBOOL(1)	T|F
CKA_VALUE_LEN	CK_ULONG	Length in bytes of value array
CKA_VALUE	Byte array	Key value (8|16|24)

2.4 Slot and Token Management

Each slot handled by LSM-PKCS11 daemon is managed in memory with a dedicated descriptor, split in two parts, one for the slot and one for the embedded token on the related SecureBox. The layout of this descriptor is shown in the following table.

SlotId	logical slot identifier (1..4)
Name	logical slot/token name
SlotInfo	slot information (adhering PKCS#11 CK_SLOT_INFO structure)
Token	token descriptor

The layout of token descriptor is shown in the following table.

Name	logical token name
TokenInfo	token information (adhering PKCS#11 CK_TOKEN_INFO structure)
Flags	token flags for currently opened sessions (RO, RW)
Sbox	logical descriptor
SBoxFileName	SecureBox file name
SBoxKeyFName	SecureBox protection key file name
SBoxName	logical name of SecureBox
SBoxKey	SecureBox protection key
NumObjs	number of objects held on token
ObjSet	array of object descriptors

2.5 OpenSSL Interface

To perform cryptographic operations the LSM daemon uses the OpenSSL crypto library (libeay). This choice has been suggested by some considerations:

- the main objective of LSM-PKCS11 is the development of a Secure Module (software or hardware), not the cryptographic operations performed by it, just leaving to a dedicated project this task;
- among many possibilities offered by the open source scenario, OpenSSL is the more frequently used, the more tested, the more reliable;
- the interface of OpenSSL is quite difficult, but almost any cryptographic library is difficult to use: but OpenSSL is widely used, and many examples exist for a proper integration[5].

[5] Actually, for the development of LSM-PKCS11, many suggestions have been taken from 'gpkcs11' (GNU pkcs11), an open source project developed by Lutz Behnke (behnke@trustcenter.de), with the support of Ron Ramsay

The interface to OpenSSL library is implemented in a dedicated module, in which a set of subroutines performs the required cryptographic tasks with the support of OpenSSL functions and its internal data types (RSA, DSA, des_cblock, MD2_CTX, MD5_CTX, SHA_CTX, etc.). This module maps data from OpenSSL format to a format internal to LSM, more adequate to PKCS#11 standard interface.

2.6 Slots, Tokens, Objects, Sessions

The global design of LSM-PKCS11 daemon management for slots, tokens, objects and sessions is reassumed in figure 2.

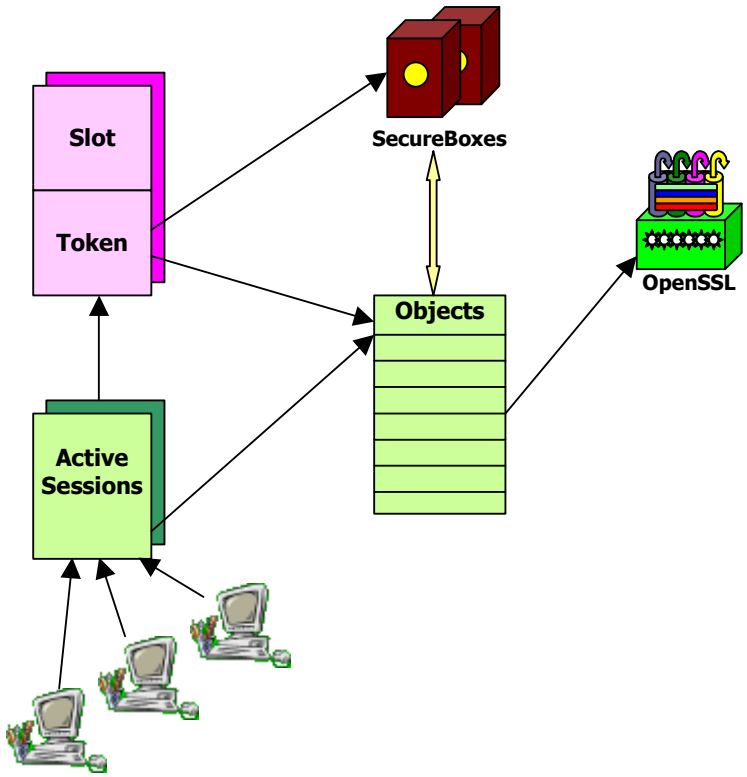


Figure 2 – LSM-PKCS11 Slots, Tokens, Objects, Sessions management

(Ron.Ramsay@OpenDirectory.com.au), Paul Augustyn (gustaw@cryptotech.com.pl), Jean-Marc Desperrier (jean-marc.desperrier@certplus.com), Régis Gaschy (gaschy@stud.fh-frankfurt.de), Greg Robert (robert@stud.fh-frankfurt.de).

3. Project Architecture

LSM-PKCS11 project is organized is a set of modules, as detailed in figure 3.

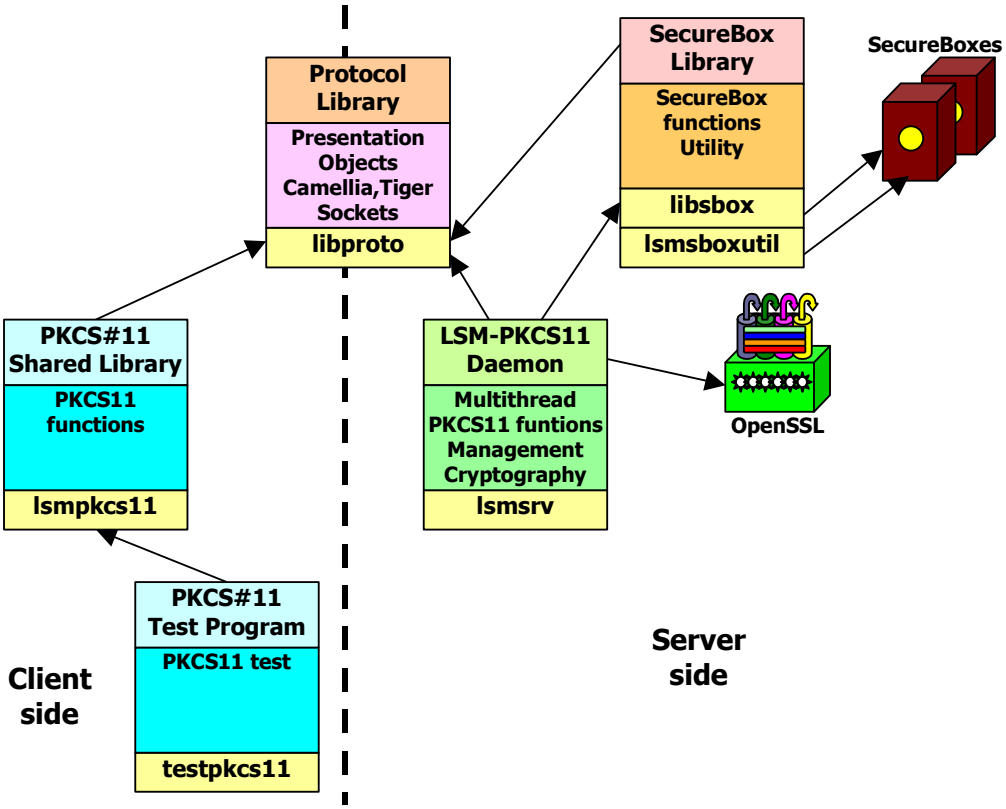


Figure 3 – LSM-PKCS11 Project modules

All the modules are platform independent, and run both in Linux and Microsoft Windows environment. The current version has been developed and tested on the following platforms:

Linux Fedora 4, 64 bits
Linux Fedora 4, 32 bits
MS Windows 2000/XP, 32 bits

There should be no problems in porting the modules in the other Unix environment common to OpenSSL distributions. A TODO of this project is the integration with 'autoconf' utility: currently the compilation support for Linux is at the minimum level, but works without any problem.

For the MS Windows environment full project support is present for Visual Studio 6.0 (all .dsw and .dsp project files are present in the distribution).

3.1 Protocol Library module

This module contains the following sources:

- lsmprolib.c, lsmprolib.h — basic protocol and socket management
- lsmproto.c, lsmproto.h — network serialization/deserialization
- pkcs11proto.c, pkcs11proto.h — PKCS11 serialization/deserialization
- lsmobjects.c, lsmobjects.h — object and attributes handling
- lsmcamellia.c, lsmcamellia.h — Camellia cryptography
- lsmtiger.c, lsmtiger.h — Tiger hashing
- minilzo.c, lsmlzo.c, minilzo.h, lsmlzo.h, lzoconf.h — network compression

3.2 SecureBox Library module

This module contains the following sources:

- lsmsbox.c, lsmsbox.h — SecureBox handling
- lsmsboxutil.c — SecureBox utility program
- lsmsboxtest.c — SecureBox test program

3.3 LSM-PKCS11 Daemon module

This module contains the following sources:

- lsmsrv_main.c — main thread and process management
- lsmsrv_child.c — child thread management
- lsmsrv_funcs.c — PKCS11 functions
- lsmsrv_lsm.c — LSM-PKCS11 management functions
- lsmsrv_crypto.c — cryptographic handling
- lsmsrv_conf.c — configuration file handling
- lsmsrv_queue.c — dynamic memory queues and pools handling
- lsmsrv_lib.c — general purpose internal functions
- lsmsrv_uxth.c — Unix multithread functions
- lsmsrv_win32.c — MS Windows multithread functions
- lsmsrv_log.c — log functions
- lsmsrv_vars.c — daemon global variables
- lsmsrv.h — daemon internal header

3.4 PKCS#11 Shared Library module

This module contains the following sources:

- pkcs11.c — PKCS#11 shared library functions
  pkcs11.h
  pkcs11types.h
  pkcs11proto.h
- pkcs11net.c — client network handling
- pkcs11net.h
- pkcs11util.c — internal functions
  pkcs11util.h
  pkcs11int.h
- pkcs11uxtht.c — Unix multithread functions
- pkcs11win32.c — MS Windows multithread functions

3.5 PKCS#11 Test Program module

This module contains the following sources:

- testpkcs11.c — PKCS#11 test program

4. Building LSM-PKCS11

The building process for LSM-PKCS11 modules is quite simple.

4.1 Building Linux environment

Untar the distribution file LSM-PKCS11.tar.gz under any directory, and proceed as follows:

For 64 bits environment:

```
rm Makefile
ln –s Makefile.64 Makefile
make
make install
```

For 32 bits environment:

```
rm Makefile
ln –s Makefile.32 Makefile
make
make install
```

Then create the following directories:

```
mkdir /usr/local/lsm
mkdir /usr/local/lsm/conf
mkdir /usr/local/lsm/keys
mkdir /usr/local/lsm/log
mkdir /usr/local/lsm/sbox
```

4.2 Building MS Windows environment

Unzipo the distribution file LSM-PKCS11.zip under any directory, and proceed as follows:

```
Run Visual Studio 6.0
Open the project lsm.dws (it is under the main directory LSM-PKCS11)
Build, batch-build, rebuild all
```

Then create the following directories:

```
mkdir c:\usr\local\lsm
mkdir c:\usr\local\lsm\conf
mkdir c:\usr\local\lsm\keys
mkdir c:\usr\local\lsm\log
mkdir c:\usr\local\lsm\sbox
```

5. Using LSM-PKCS11

Now it is time to try to use your new LSM-PKCS11 tool. In this chapter I try to explain in few and easy words how to proceed. Two different procedures are described, one for Linux environment and one for MS Windows.

5.1 Running in Linux environment

Prepare your SecureBoxes, as detailed in paragraph 5.3, and copy them under the directory /usr/local/lsm/sbox, and the related protection key files under the directory /usr/local/lsm/keys.

Then prepare a configuration file for LSM-PKCS11 daemon, as detailed in paragraph 5.4, and copy it under the directory /usr/local/lsm/conf.

At this point you may run the daemon issuing the command:

```
/usr/local/bin/lsmsrv –c /usr/local/lsm/conf/<your-lsmsrv.cfg> -d [-s]
```

The daemon is ready to be used.

Now edit your file /etc/ld.conf.so, and if missing add to it the entry:

```
/usr/local/lib
```

and, as root, run the system command:

```
ldconfig
```

so that the shared library /usr/local/lib/liblsmpkcs11.so.1.0.1 is correctly mapped in the system.

Then prepare the configuration file 'lsmpkcs11.cfg' for LSM-PKCS11 shared library, as detailed in paragraph 5.4, and copy it under the directory /usr/local/lsm/conf.

Now you may proceed to use the library and the daemon, for example with the command:

```
/usr/local/bin/testpkcs11 [<RSA-key-size>]
```

where <RSA-key-size> may take the values 512, 1024, 2048, 4096 (default 1024).

5.2 Running in MS Windows environment

Prepare your SecureBoxes, as detailed in paragraph 5.3, and copy them under the directory c:\usr\local\lsm\sbox, and the related protection key files under the directory c:\usr\local\lsm\keys.

Then prepare a configuration file for LSM-PKCS11 daemon, as detailed in paragraph 5.4, and copy it under the directory c:\usr\local\lsm\conf.

At this point you may run the daemon. Create the directory:

c:\usr\local\lsm\bin

and copy under it the daemon executable file lsmsrv.exe (from the directory bi\d or bin\r of your development directory). Then, under a DOS window, issue the command:

c:\usr\local\lsm\bin\lsmsrv.exe –c c:\usr\local\lsm\conf\<your-lsmsrv.cfg> [-s]

The daemon is ready to be used.

Now copy the LSM-PKCS11 shared library (lsmpkcs11.dll) to the directory 'system32' of your system (i.e. c:\WINNT\system32), so that it can be found by application programs.

Then prepare the configuration file 'lsmpkcs11.cfg' for LSM-PKCS11 shared library, as detailed in paragraph 5.4, and copy it under the directory c:\usr\local\lsm\conf.

Now you may proceed to use the library and the daemon, for example with the command:

testpkcs11.exe [<RSA-key-size>]

where <RSA-key-size> may take the values 512, 1024, 2048, 4096 (default 1024).

5.3 Handling SecureBoxes

To prepare a SecureBox use the 'lsmsboxutil' command (or 'lsmsboxutil.exe' unde MS Windows). This program has the following syntax:

lsmsboxutil <sbox-file> <KEY> <command> <parameter>

where:

<sbox-file>	file containing the box (suggested extension .sbx)
<KEY>	current protection key for the box

and allowed commands are:

-c <sbox-name>	create a new empty box
-k <sbox-name> <new-KEY>	change the protection key to <new-KEY>
-K <sbox-name> <key-file-name>	save the protection key to a file (extension .key)
-s <sbox-name> <new-SOPIN>	change SO (Security Officer) PIN
-u <sbox-name> <new-USPIN>	change user PIN
-r <sbox-name>	rebuild the box, garbaging data space
-d <dump-file>	dump box content in a readable form

Thus, for example, to make a new SecureBox and save its protection key in a file:

lsmsboxutil /usr/local/lsm/sbox/sboxtest1.sbx 12345678 -c SBOXTEST1

and

lsmsboxutil /usr/local/lsm/sbox/sboxtest1.sbx 12345678
-K SBOXTEST1 /usr/local/lsm/keys/sboxtest1.key

5.3 Configuring LSM-PKCS11

To run LSM-PKCS11 you need two configuration files, one for the daemon and one for the shared library.

At run time, the daemon configuration file may reside wherever you like and with the name you like, as it is referenced by the option '-f' of 'lsmsrv' program. Instead the shared library configuration file must reside under the directory /usr/local/lsm/conf (c:\usr\local\lsm\conf in MS Windows), and its name must be 'lsmpkcs11.cfg'.

The daemon configuration file is a text file, where each line identifies a parameter and its value in the form:

<parameter-name> <parameter-value>

unless it is empty or begins with '#' comment character. Required parameters are:

RootPath	where is LSM home
SboxPath	where LSM stores secure boxes
LogPath	where LSM makes logs
EvnlogFile	log file name
EvnLogLevel	warning | normal | error | debug
ErrLogFile	errlog file name
ErrLogLevel	warning | normal | error | debug
ListenOn	listening port
Resolve	yes or no (resolve remote name)
Slot	slot name
SlotSBox	slot SBox file name
SboxName	slot SBox name
SboxKeyFile	slot SBox protection key file name

where the slot block may be repeated up to four times. For example for two SecureBoxes (slots):

RootPath	/usr/local/lsm
SboxPath	/usr/local/lsm/sbox
LogPath	/usr/local/lsm/log
EvnlogFile	log.log
EvnLogLevel	error
ErrLogFile	err.log
ErrLogLevel	error
ListenOn	5555
Resolve	no
Slot	LSMTEST1
SlotSBox	sboxtest1.sbx
SboxName	SBOXTST1
SboxKeyFile	/usr/local/lsm/keys/sboxtest1.key
Slot	LSMTEST2
SlotSBox	sboxtest2.sbx
SboxName	SBOXTST2
SboxKeyFile	/usr/local/lsm/keys/sboxtest2.key

The shared library configuration file 'lsmpkcs11.cfg' is a text file, where each line identifies a parameter and its value in the form:

<parameter-name> = <parameter-value>

unless it is empty or begins with '#' comment character. Required parameters are:

[PKCS11]
Server = DNS name or IP address of the server running LSM-PKCS11 daemon
Service = service port on which the daemon is listening

For example:

[PKCS11]
Server = lsm.clizio.home
Service = 5555