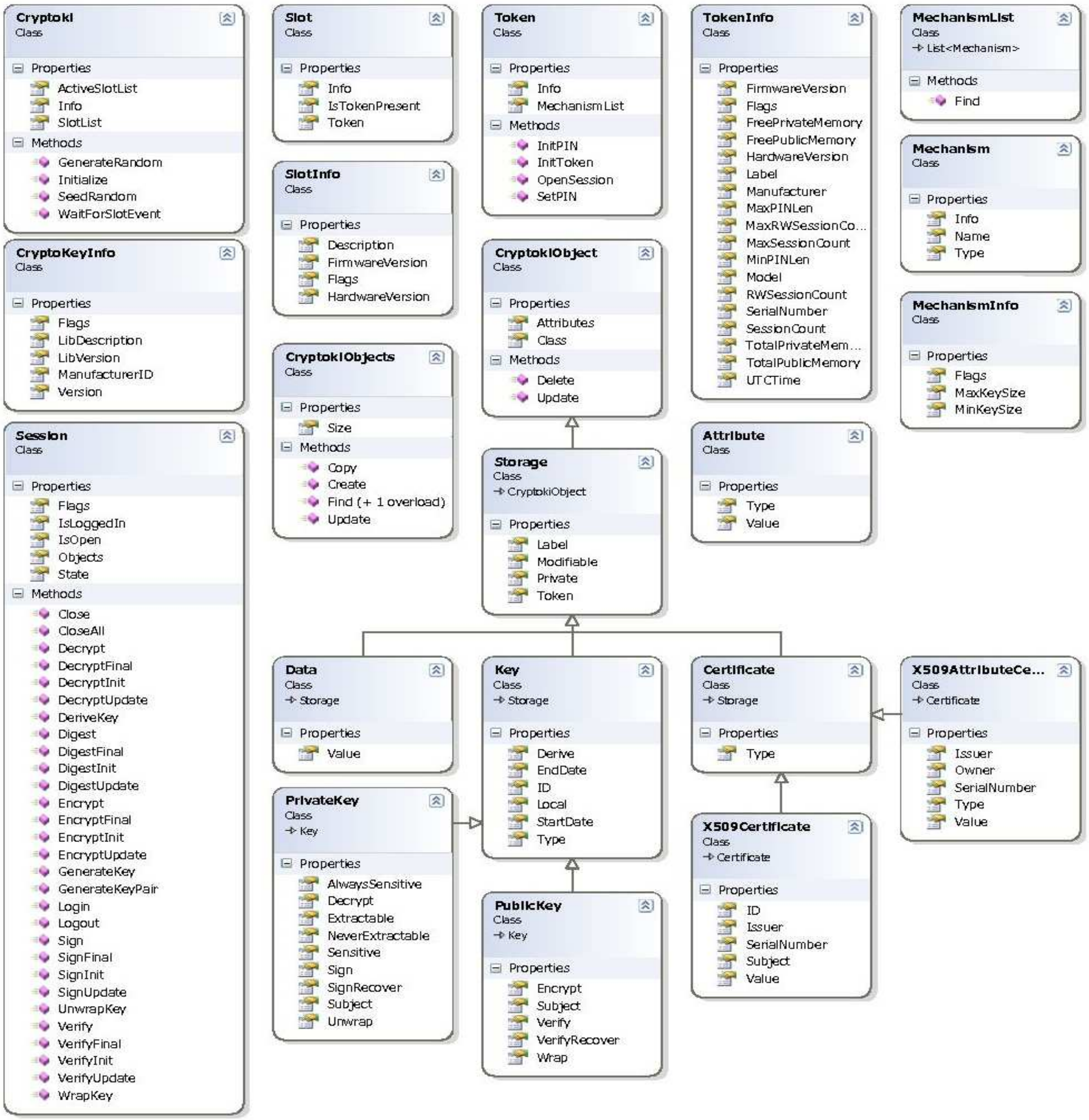
.NET PKCS#11 – NCryptoki

Data: 10/12/2007, Author: Ugo Chirico – http://www.ugosweb.com

PKCS#11 functions for accessing smart cards is contained in a specific dynamic link library (dll) which is implemented in C and thus it is unmanaged code. Therefore to use Cryptoki functions directly in your C# and/or VB.NET code there is only one chance: using platform invoke services, such as the System.Runtime.InteropServices namespace and the CLR supplied by the framework to import the unmanaged functions exported by Cryptoki library in your C# and/or VB.NET managed code. Platform invoke enables managed code to call functions exported from an unmanaged dll such as Win32 API and custom dlls while the CLR handles dll loading and parameter marshaling.
Howewer importing functions from an unmanaged dll, expecially from Cryptoki dll, requires some advanced skills in C/C++ and .NET and compels a lot of tedious work in declaring the prototypes related to the functions to import and in dealing with custom parameters marshalling.
In order to avoid from dealing with unmanaged code and avoiding also from many tedious work, we are developing a C++ managed extension package of classes which maps Cryptoki functions by an high-level set of classes usable from C# and VB.NET. Next Figures show the class hierarchy of .NET Cryptoki.



The programming paradigm is almost similar to native code implementations. *Cryptoki* functions are mapped into specific classes outlining the classification as reported in the PKCS#11 specifications.
In depth, Cryptoki is the main class whitch manages the entire library, Slot and Token classes enclose *slot*-handling and *token*-handling functions, while the CryptokiObject class encapsulates object-handling functions as well as definitions about objects' classes and their attributes. Finally, Session class includes the session-opening and session-closing functions, the login function, the list of all objects, cryptographic and hashing functions, etc
Loading and initializing the native *Cryptoki* library are chores of the static method Cryptoki.Initialize which takes as parameter the *path* of the native library. The following paragraphs show an example in C# and in VB.NET which compute a digital signature over the string "Hello World".
.NET Cryptoki, aka NCryptoki, can be downloaded from the author's web site: http://www.ugosweb.com
After downloading you must install it by clicking on the setup executable.
The following are examples about working with .NET Cryptoki in C# and VB.NET. However, on the author's web site, http://www.ugosweb.com, you can find many other samples and manuals about NCryptoki.

C#

The following is an example in C# which computes a digital signature over the string "Hello World" using NCryptoki API:

```
import USCToolkit.NCryptoki;
….

// Creates a Cryptoki object related to the specific PKCS#11 native library
Cryptoki cryptoki = new Cryptoki("SIMPKCS11.dll");

cryptoki.Initialize();

// Prints all information relating to the native library
CryptokiInfo info = cryptoki.Info;
Console.WriteLine(info.Version);
Console.WriteLine(info.ManufacturerID);
Console.WriteLine(info.LibDescription);

// Reads the set of slots containing a token
SlotList tokenslots = cryptoki.ActiveSlots;
if(tokenslots.Count == 0)
    throw new Exception("No token inserted");

// Gets the first token available
Token token = tokenslots[0].Token;

// Opens a read/write serial session
Session session =
    token.OpenSession(Session.CKF_SERIAL_SESSION | Session.CKF_RW_SESSION,
                      null,
                      null);

// Executes the login passing the user PIN
session.Login(Session.CKU_USER, "1234");

// Searchs for an RSA private key object
// Sets the template with its attributes
List<ObjectAttribute> template = new List<ObjectAttribute>();
template.Add(new ObjectAttribute(ObjectAttribute.CKA_CLASS,
CryptokiObject.CKO_PRIVATE_KEY));
template.Add(new ObjectAttribute(ObjectAttribute.CKA_KEY_TYPE, Key.CKK_RSA));
template.Add(new ObjectAttribute(ObjectAttribute.CKA_LABEL, "Ugo's new Key"));

// Launchs the search specifying the template just created
List<CryptokiObject> objects = session.Objects.Find(template, 10);

for (int i = 0; i < objects.Count; i++)
{
    Console.WriteLine(((PrivateKey)objects[i]).Label);
}
```

```
RSAPrivateKey privateKey;
// If the private keys is not found generates a new key pair
if(objects.Count == 0)
{
    List<ObjectAttribute> templatePub = new List<ObjectAttribute>();
    templatePub.Add(new ObjectAttribute(ObjectAttribute.CKA_CLASS,
CryptokiObject.CKO_PUBLIC_KEY));
    templatePub.Add(new ObjectAttribute(ObjectAttribute.CKA_TOKEN, true));
    templatePub.Add(new ObjectAttribute(ObjectAttribute.CKA_PRIVATE, true));
    templatePub.Add(new ObjectAttribute(ObjectAttribute.CKA_LABEL, "Ugo's new Key"));

    List<ObjectAttribute> templatePri = new List<ObjectAttribute>();
    templatePri.Add(new ObjectAttribute(ObjectAttribute.CKA_CLASS,
CryptokiObject.CKO_PRIVATE_KEY));
    templatePri.Add(new ObjectAttribute(ObjectAttribute.CKA_TOKEN, true));
    templatePri.Add(new ObjectAttribute(ObjectAttribute.CKA_PRIVATE, true));
    templatePri.Add(new ObjectAttribute(ObjectAttribute.CKA_LABEL, "Ugo's new Key"));

    //gets the first object
    Key[] keys = session.GenerateKeyPair(Mechanism.RSA_PKCS_KEY_PAIR_GEN, templatePub,
templatePri);
    privateKey = (RSAPrivateKey)keys[1];
 }
 else
 {
    //computes the digital signature
    privateKey = (RSAPrivateKey)objects[objects.Count - 2];

    Console.WriteLine(privateKey.Label);

    string helloworld = "Hello World";
    byte[] text = Encoding.ASCII.GetBytes(helloworld);

    // Evaluates the hash using the SHA1 algorithm
    session.DigestInit(Mechanism.SHA1);

    // Here you get the digest and its length
    //toSign is the Byte vector holding the file to sign
    byte[] digest = session.Digest(text);

    // launches the digital signature operation with a RSA_PKCS mechanism
    session.SignInit(Mechanism.RSA_PKCS, privateKey);

    // computes the signature
    byte[] signature = session.Sign(digest);

    Console.Write(signature);
}

// Logouts and closes the session
session.Logout();
session.Close();
```

VB.NET

The following is an example in VB.NET which computes a digital signature over the string “Hello World” using NCryptoki API:

```
Imports USCToolkit.NCryptoki
…

Dim cryptki as Cryptoki
Dim info as CryptokiInfo
Dim tokenslots as Array(of Slot)
Dim tok As Token
Dim sess as Session
Dim template as List(of ObjectAttribute)

‘ Creates a Cryptoki object related to the specific PKCS#11 native library
Cryptoki cryptki = new Cryptoki(“CardOS_PKCS11.dll”)

cryptki.Initialize()

‘ Prints all information relating to the native library
CryptokiInfo info = cryptki.Info
Console.WriteLine(info.Version)
Console.WriteLine(info.ManufactureID)
Console.WriteLine(info.LibDescryption)

‘ Reads the set of slots containing a token
tokenslots = cryptki.ActiveSlots
if(tokenslots.Count == 0) then
   return
end if

‘ Gets the first token available
tok = tokenslots(0)

‘ Opens a read/write serial session
sess = tok.OpenSession(Session.CKF_SERIAL_SESSION | Session.CKF_RW_SESSION,
                       null,
                       null);

‘ Executes the login passing the user PIN
sess.Login(false, “1234”)

‘ Searchs for an RSA private key object
‘ Sets the template with its attributes
template = new List(of ObjectAttribute)
template.Add(new ObjectAttribute (ObjectAttribute.CKA_CLASS,
CryptokiObject.CKO_PRIVATE_KEY))
template.Add(new ObjectAttribute (ObjectAttribute.CKA_KEY_TYPE, Key.CKK_RSA))
template.Add(new ObjectAttribute (ObjectAttribute.CKA_LABEL, "Ugo's new Key"))

‘ Launchs the search specifying the template just created
Dim objects as List(of CryptokiObject)
objects = sess.Objects.Find(template);

‘ If the private keys is found computes the digital signature
if(objects.Count > 0) then

   ‘gets the first object
   privateKey = objects(0)

   string helloworld = “Hello World”
   Dim toSign() as byte
   toSign = Encoding.ASCII.GetBytes(helloworld)

   Dim digest() as byte
   Dim signedData() as byte

   ‘ Evaluates the hash using the SHA1 algorithm
   sess.DigestInit(Mechanism.SHA_1, null)

   ‘ Here you get the digest
    digest = sess.Digest(toSign)

   ‘ launches the digital signature operation with a RSA_PKCS mechanism
   sess.SignInit(Mechanism.RSA_PKCS, privateKey)

   ‘ computes the signature
   signedData = sess.Sign(digest)

End if

‘ Logouts and closes the session
```

```
sess.Logout()
sess.Close()
```