Functional Documentation for "NFC CSP Light"

Version 1.0

Prepared by: "Vincent Le Toux"

Date: 03/02/2014

Table of Contents

Table of Contents

Revision History

Description 4

System Specifications 4

Hardware 4

External System Dependencies 4

Architecture overview 4

Files 5

Virtual smart card 6

    Data storage 6

    Cryptography 6

    PIN attempts check 7

    Limitations 7

    PIN Caching 7

Quality tests 8

    csptestsuite 8

Revision History

This section records the change history of this document.

Name	**Date**	**Reason For Changes**	**Version**
Vincent Le Toux	03/02/2014	Creation	1.0

Description

NFC CSP is a software used to emulate the cryptographic functions of smart card using a NFC tag connected to a PCSC reader. It has been designed allow a quick and secure login, independently of the password, like in a hospital. This software is not suitable to provide a very high level of security.

The software provide added cryptographic capabilities to be used on Windows. The software was design to support all the common smart card scenarios like :

- Smart card logon using Active Directory
- Smart card logon using EIDAuthenticate
- SSL Authentication using Internet Explorer (or any CAPI compliant browser)
- Digital Signature of emails using Outlook
- Digital Signature of documents using Word
- Encryption of documents using EFS or Bitlocker

System Specifications

Operating system supported are :

- Windows XP, Vista, Seven, 8
- Windows Server 2003, 2008, 2012

Hardware

The smart card reader must be a PCSC compliant NFC reader and support the pseudo APDU for reading the NFC ID (FFCA000000).

External System Dependencies

This software depends on certain external systems to complete some or all of its tasks. This section lists those dependencies explicitly:

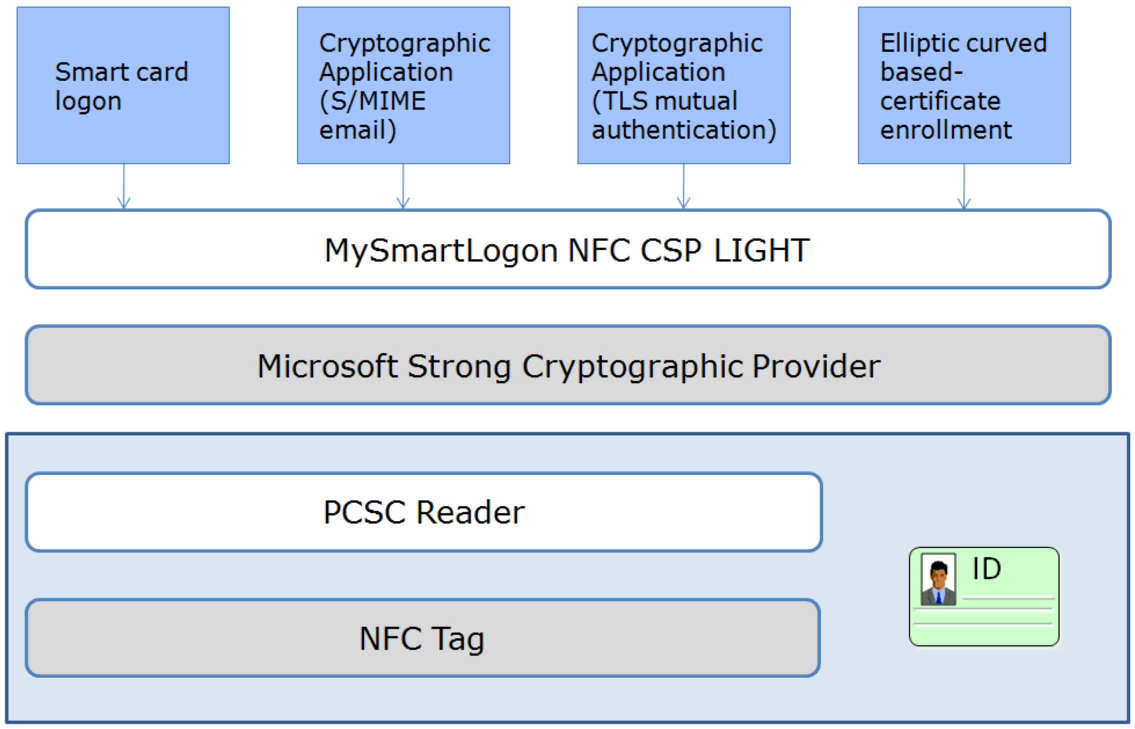
External System	Dependencies on that System
Windows Installer 3.0	None

Architecture overview

There are two ways possible to emulate a smart card in Windows. The first one is to write a full CSP (Cryptographic Service Provider), and the second one is to implement only a subset, named minidriver, by using the existing CSP named "Base Smart card CSP".

Implementing a virtual smart card with the minidriver model requires to compute raw RSA operations and to emulate a file system. Implementing a CSP requires only to wrap the calls to the classic RSA provider and to protect the cryptographic material which is easier than building a minidriver.

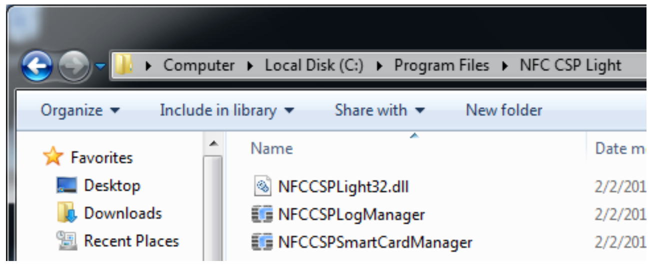
The implementation of a virtual smart card reader is not needed as using a PCSC reader is a requirement. However this solution requires that the files are signed every time the CSP is built, like a kernel mode driver.


Smart card logon
Cryptographic Application (S/MIME email)
Cryptographic Application (TLS mutual authentication)
Elliptic curved based-certificate enrollment
MySmartLogon NFC CSP LIGHT
Microsoft Strong Cryptographic Provider
PCSC Reader
NFC Tag
ID

Files

NFC CSP is made of the following files :

- NFCCSPLight[32|64].dll which is a Crytographic Service Provider, a dll run by the CAPI library.
- NFCCSPSmartManager.exe which is a program design to manage the certificates and the cryptographic container
- NFCCSPLogManager.exe which is a utility used to get diagnostic traces



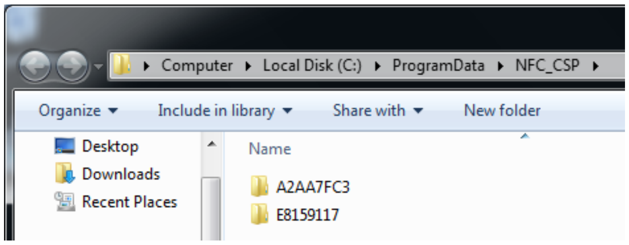
Name	Date m
NFCCSPLight32.dll	2/2/201
NFCCSPLogManager	2/2/201
NFCCSPSmartCardManager	2/2/201

Virtual smart card

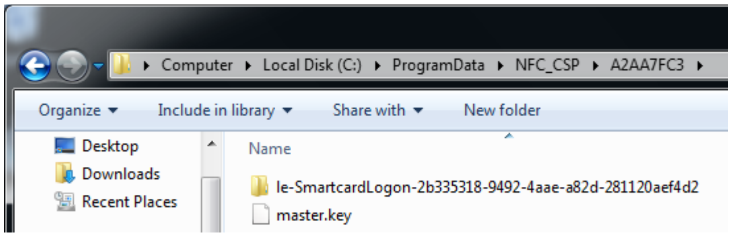
Data storage

The data is stored in the Common Application Folder, located on Windows 7 on c:\ProgramData. The folder used by the application is NFC_CSP.

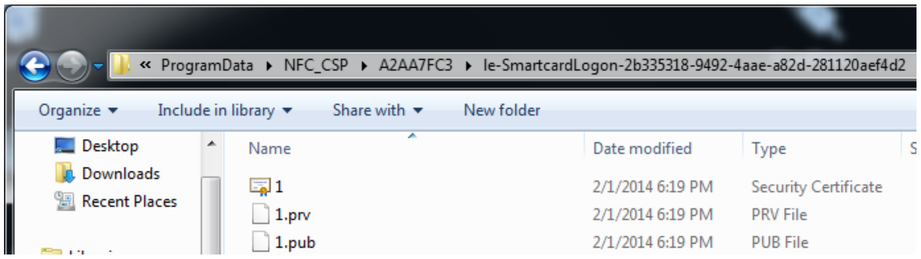
There are inside this folder a directory per smart card emulated. The name of the directory is computed with a hash of the characteristics of the smart card.



Inside the virtual smart card, there are a directory per container and a master key file. The masterkey file is encrypted using the characteristics of the smart card along with the PIN if it is defined.


Computer ▸ Local Disk (C:) ▸ ProgramData ▸ NFC_CSP ▸ A2AA7FC3 ▸
Organize
Include in library
Share with
New folder
Desktop
Downloads
Recent Places
Name
le-SmartcardLogon-2b335318-9492-4aae-a82d-281120aef4d2
master.key

Each container can contains up to two public key pair (signature key and key exchange key) as specified by the CAPI interface. For each key, the public key and the private key is written in two different files with different cryptographic key, allowing to read the public key without having being authenticated. A certificate can be optionally stored encrypted.


« ProgramData ▸ NFC_CSP ▸ A2AA7FC3 ▸ le-SmartcardLogon-2b335318-9492-4aae-a82d-281120aef4d2
Organize
Include in library
Share with
New folder
Desktop
Downloads
Recent Places

Name	Date modified	Type
1	2/1/2014 6:19 PM	Security Certificate
1.prv	2/1/2014 6:19 PM	PRV File
1.pub	2/1/2014 6:19 PM	PUB File

Cryptography

There are two keys derived from the NFC tag.

The first one is used to access public material. It is based on the characteristics of the tag like its ATR or its UID.

The second one is used to access the private material. It is based on the same basic characteristics than the public key with the PIN as an added secret. Several PKDF2 iterations to provide brute force attempts. One derivation takes approximately one half of a second.

These key are then used as entropy to the CryptProtectData function which computes a key unique to the local machine. As a consequence, the data cannot be transferred to another computer.

Here is an assessment to access the private key. With an UID of 5 bytes, the time needed to test all the combination is 256^5/2 = 549755813888 s = 17 432 years. With an UID of only 3 bytes, the time drops to 0,5 year. To increase this time, a PIN can be set. Each digit set multiplies the time per 10 and each letter/digit set multiplies the time per 36.

The PIN is used as a mitigation to the risk of a lost NFC tag.

Here is an estimation of the time required to brute force offline the PIN, having access to the computer where is stored the data :

Password strength	Time
4 number digits	1 hour
8 number digits	19 months
4 letters/numbers digits	3 months
4 digits with symbols	1 year, 4 months
8 digits with symbols	124 millions years

PIN attempts check

The PIN attempts cannot be stored in the secure container because the container could be saved before the attempt and erased after the login attempt or simply write protected.

As a consequence, the PIN attempts are not monitored.

Limitations

The following table lists the limitations enforced in the product

Limitation	Comment
Maximum container length	100 characters
Minimum PIN size	0 characters
Maxmimum PIN size	255 characters
Digits forbidden in the container name	< > : \ " / \ | ? *
Digits forbidden in the PIN	None

PIN Caching

The PIN is cached on a process level as advised by Microsoft.

Each time the PIN needs to be accessed, it is securely erased from the memory.

```
SecureZeroMemory(szPin, sizeof(szPin));
```

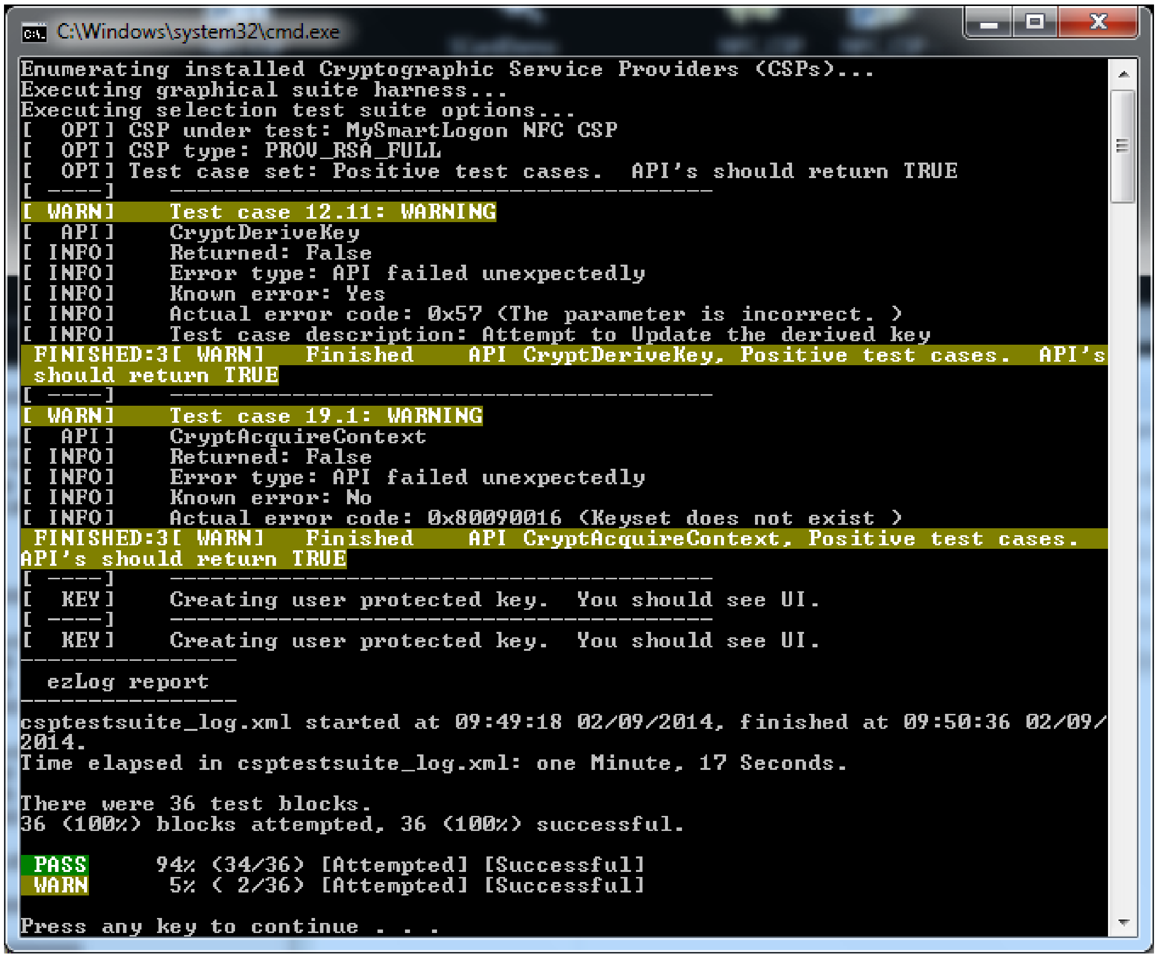
The PIN is stored in the cache encrypted.

```
RtlEncryptMemory((*it)->encryptedPin, MAX_PIN_SIZE, 0);
```

Quality tests

csptestsuite

NFC CSP Light passes successfully the Microsoft test suite designed for CSP.



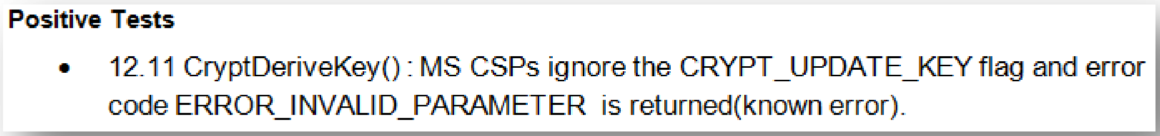
```
Enumerating installed Cryptographic Service Providers (CSPs)...
Executing graphical suite harness...
Executing selection test suite options...
[  OPT] CSP under test: MySmartLogon NFC CSP
[  OPT] CSP type: PROV_RSA_FULL
[  OPT] Test case set: Positive test cases.  API's should return TRUE
[ ----]     ----------------------------------------
[ WARN]     Test case 12.11: WARNING
[  API]     CryptDeriveKey
[ INFO]     Returned: False
[ INFO]     Error type: API failed unexpectedly
[ INFO]     Known error: Yes
[ INFO]     Actual error code: 0x57 (The parameter is incorrect. )
[ INFO]     Test case description: Attempt to Update the derived key
 FINISHED:3[ WARN]   Finished    API CryptDeriveKey, Positive test cases.  API's
 should return TRUE
[ ----]     ----------------------------------------
[ WARN]     Test case 19.1: WARNING
[  API]     CryptAcquireContext
[ INFO]     Returned: False
[ INFO]     Error type: API failed unexpectedly
[ INFO]     Known error: No
[ INFO]     Actual error code: 0x80090016 (Keyset does not exist )
 FINISHED:3[ WARN]   Finished    API CryptAcquireContext, Positive test cases.
API's should return TRUE
[ ----]     ----------------------------------------
[  KEY]     Creating user protected key.  You should see UI.
[ ----]     ----------------------------------------
[  KEY]     Creating user protected key.  You should see UI.
----------------
  ezLog report
----------------
csptestsuite_log.xml started at 09:49:18 02/09/2014, finished at 09:50:36 02/09/
2014.
Time elapsed in csptestsuite_log.xml: one Minute, 17 Seconds.

There were 36 test blocks.
36 (100%) blocks attempted, 36 (100%) successful.

 PASS      94% (34/36) [Attempted] [Successful]
 WARN       5% ( 2/36) [Attempted] [Successful]

Press any key to continue . . .
```

The tests return two warnings. The first warning are caused by the underlying Microsoft CSP. This is known problem and documented as such by Microsoft (file readme.doc).



Positive Tests

- 12.11 CryptDeriveKey() : MS CSPs ignore the CRYPT_UPDATE_KEY flag and error code ERROR_INVALID_PARAMETER is returned(known error).

The second warning is triggered when all containers are removed from the smart card : there is no more container to open.