Functional Documentation for EIDVirtual

Version 2.0

Prepared by: "Vincent Le Toux"

Date: 17/05/2026

MySmartLogon

Table of Contents

Table of Contents

Revision History

Description 4

System Specifications 4

Hardware 4

External System Dependencies 4

Architecture overview 5

Files 6

Virtual smart card 7

- Answer To Reset (ATR) 7
- Data storage 7
- PIN attempts check 8
- Admin Key mechanism 8
- Virtual Smart Card limitation 8

Revision History

This section records the change history of this document.

Name	Date	Reason For Changes	Version
Vincent Le Toux	21/02/2013	Creation	1.0
Thierry Martin	28/03/2013	Update	1.1
Frédéric Bourgeois	04/04/2013	Update	1.2
Vincent LE TOUX	17/05/2026	Revision	2.0

Description

EIDVirtual is a software used to emulate a smart card using a removable device (e.g. USB key). The software was designed to support all common smart card scenarios:

- 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

EIDVirtual is using a UMDF version 2 driver to install a virtual smart card reader.

The virtual smart card is implemented using the GIDS specifications.

The following operating systems are supported:

- Windows 10, 11 and later
- Windows 2016,2019,2022,2026 and later

Hardware

The removable device must be accessed on a read/write file system. This excludes CD-ROM (DVD-ROM, bluray, …) and read protected device. Memory cards or USB keys can be used.

.

External System Dependencies

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

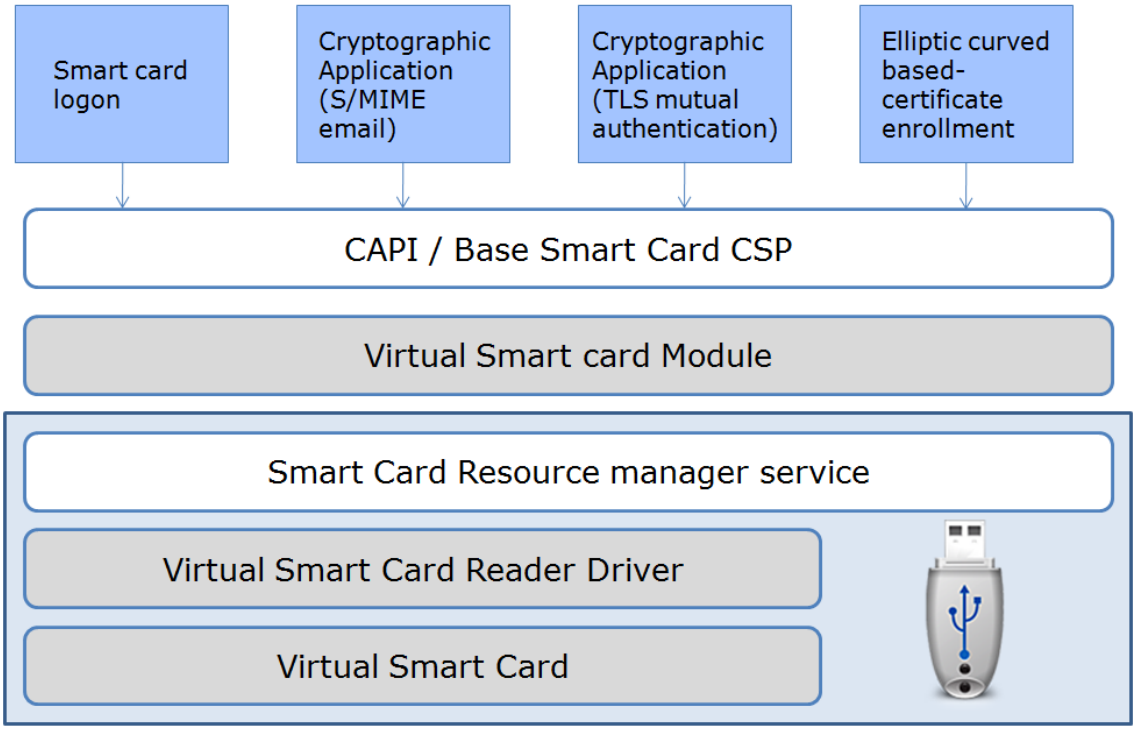
External System	Dependencies on that System
Windows Installer 3.0	None
CNG	Windows Vista
UMDF Framework 2	Windows 10 or later

Architecture overview

There are two ways 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 by using the existing CSP named "Base Smart card CSP". However, the Base Smart Card CSP requires a handle to the smart card resource manager, which in turn requires a virtual smart card reader. A virtual smart card reader requires a driver, written either using KMDF (kernel mode) or UMDF (user mode).

Because the smart card logon monitors smart card insertion and removal, writing only a CSP was not a solution. That's why it was decided to use the "Base Smart Card CSP" and to write 2 drivers :

- The virtual smart card minidriver for the Base CSP
- The virtual smart card reader for the smart card resource manager.



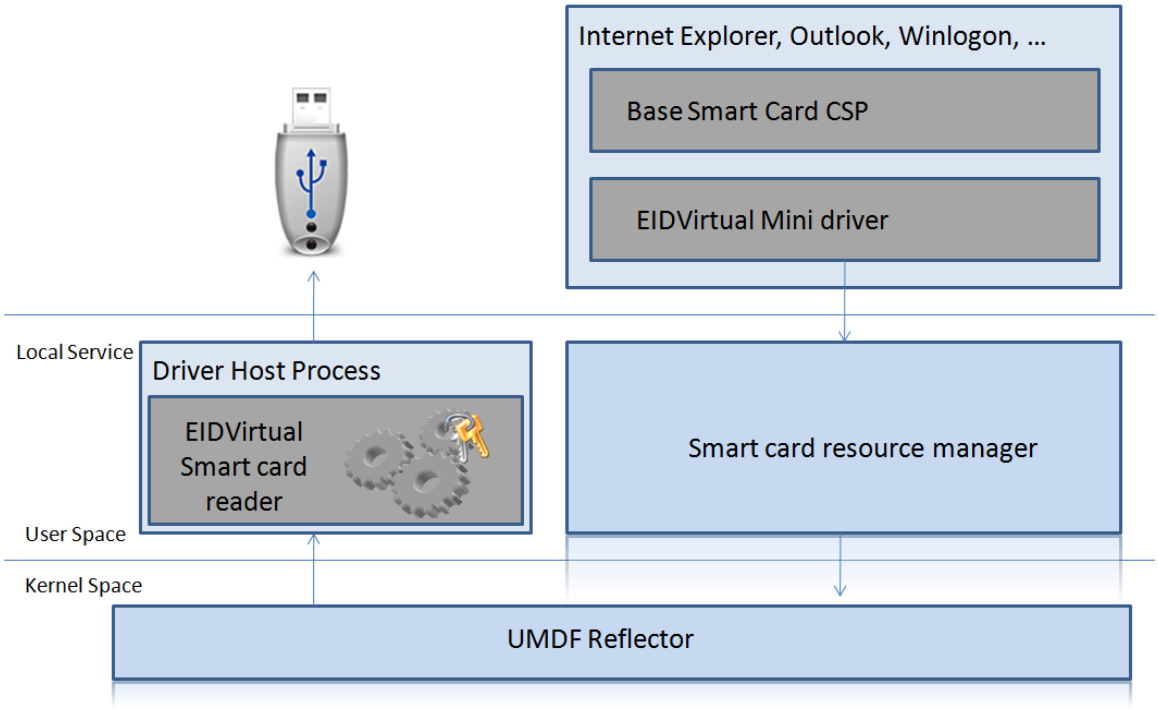
Since the minidriver runs under the current application context, cryptographic operations are performed inside the virtual smart card reader driver for security reasons.

The smart card reader driver is a UMDF driver. It runs as a service under the UMDF driver host process, using the local system account. All IOCTL commands must complete within one minute; otherwise, the UMDF driver host process is terminated. This makes live reverse engineering difficult and improves the overall security of the solution.

The choice of the UMDF framework ensures that a BUG in the virtual smart card reader won't produce a crash : in case of failures, the UMDF process is restarted.

The IOCTL sent by the smart card resource manager go into the kernel space and is redirected to the UMDF driver host process by a system driver named "UMDF Reflector". This component is part of the Windows UMDF framework.

The following diagram describes the middleware and the associated security contexts between the end application and the USB key.



Files

EIDVirtual consists of the following files:

- EIDVirtualSmartCardReader.dll which is a UMDF driver simulating a smart card reader ("EIDVirtual Smart card reader")
- EIDVirtualWizard.exe which is the program design to "format" a new smart card
- EIDVirtualManager.exe which is a program designed to manage certificates and cryptographic containers

Virtual smart card

Answer To Reset (ATR)

An ATR (Answer To Reset) is a byte sequence that uniquely identifies a type of smart card. It is used by the operating system to associate a smart card with its driver.

The ATR of the virtual smart card is: 3B 8C 01 4D 79 53 6D 61 72 74 4C 6F 67 6F 6E A5

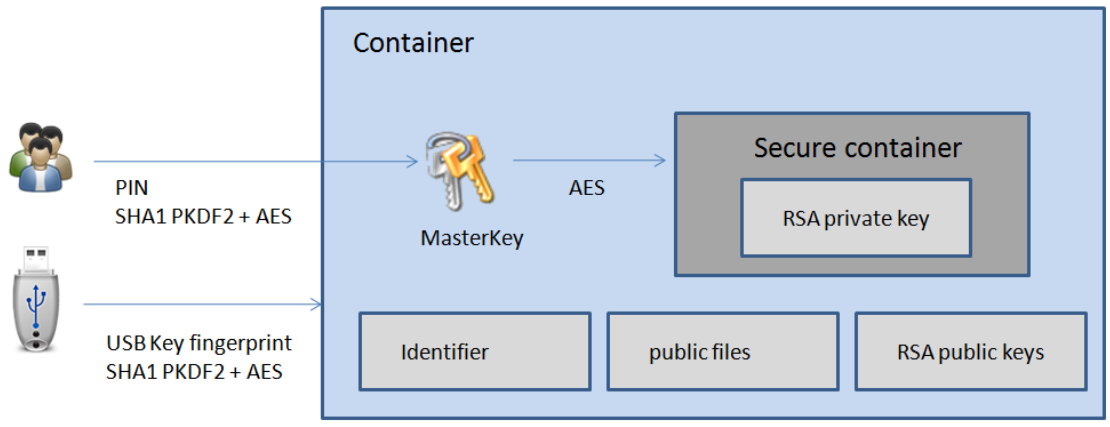
The technical characteristics of this sequence are:

TS = 0x3B	**Direct Convention**
T0 = 0x8C	**Y(1): b1000, K: 12 (historical bytes)**
TD(1) = 0x01	**Y(i+1) = b0000, Protocol T=1**
Historical bytes	**4D 79 53 6D 61 72 74 4C 6F 67 6F 6E (MySmartLogon)**
TCK = 0xA5	**checksum**

This defines the virtual smart card as using the T=1 communication protocol.

Data storage

The container is encrypted with AES-256 using a key derived from a USB key fingerprint. A copy of the container to another device won't be read by the virtual reader because the fingerprint of the key won't match the key used to encrypt the container. The key derivation mechanism (SHA-2 PBKDF2 with 4096 rounds) makes brute-force attacks impractical.



RSA private keys are protected by a master key.

PIN attempts check

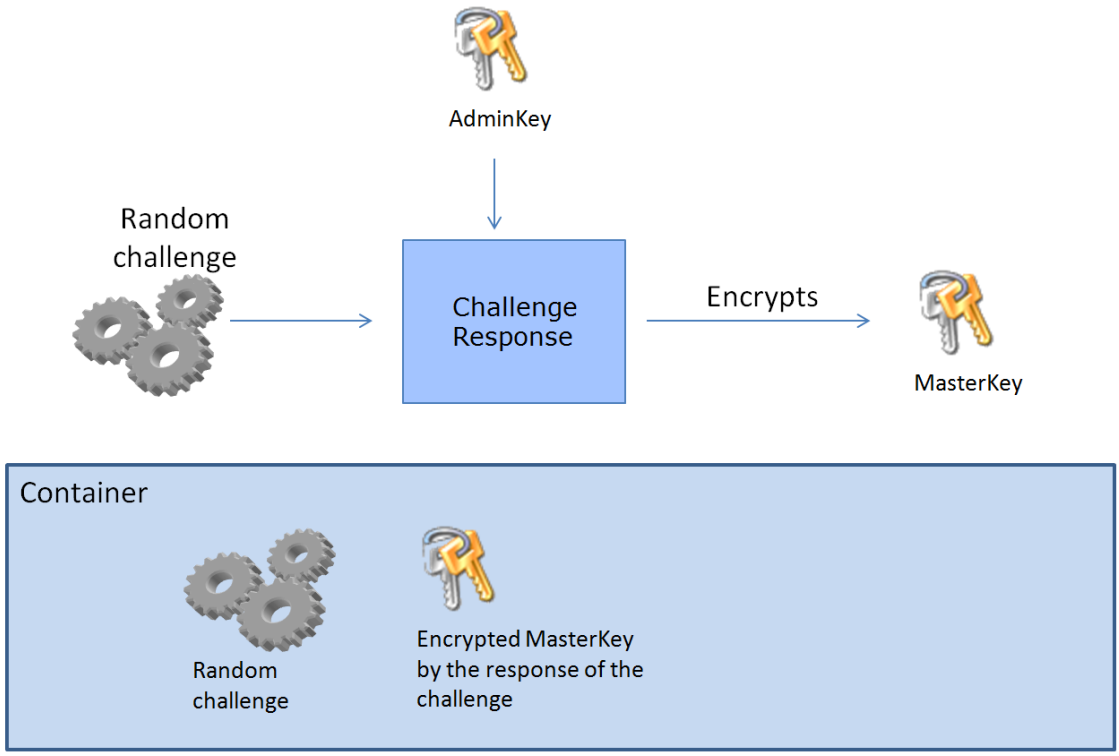
PIN attempts are stored in the secure container, but please note that the container could be backed up before an attempt and restored afterwards, or could be write-protected.

Furthermore, the user PIN can only be unblocked by the Admin PIN, which also prevents deletion of the container.

Attempt counts are stored by the smart card reader.

Admin Key mechanism

Because the container is not hardware-protected, the design ensures that the admin key is never stored inside the secure container.



Even if the master key or the challenge response is known, the admin key cannot be recovered. However, this design is vulnerable to replay attacks. That's why we are not recommending to use an Admin Key.

Virtual Smart Card limitation

The following table defines the limitations of the virtual smart card as imposed by the smart card minidriver specification more the GIDS specification.

Functionality	Limitation
Maximum number of files	65536
Maximum file size	65536 bytes
Maximum number of container	30
Maximum number of root certificate stored	50