Project	EID	
Subject	Belgian Electronic Identity Card Middleware Programmers Guide	
Version	V1.4	
Date	19/03/2003	
From	Zetes	Johan Rommelaere
Nr of Pages	16	

I Document Control

I.1 Issue/Review Cycle

Dit document wordt uitgegeven door Zetes, en bij fundamentele wijziging ter review voorgelegd aan de betrokkenen.

I.2 Distribution List

Organisation	Name
Zetes	Johan Rommelaere
	Alex Driesen
	Bart Symons
	Patrick Andries
FedICT	Bart Sijnave
RRN	Daniel Pedoux
Steria	Guy Kivits

I.3 Version Log

Version	Date	Prepared By	Comments
1.00	13/12/2002	Zetes / P Andries	Initiated the document
		Zetes/J Rommelaere	Reviewed
		Zetes/B Symons	Reviewed
1.10	14/1/2003	Zetes / P. Andries	Updated the document
1.30	04/03/2003	Zetes/P.Andries	Changed the documentation to the latest state of the software
1.40	19/03/03	Zetes/P.Andries	Added documentation for the CryptExportKey function

I.4 Changes since previous issues

I.4.1 Changes in

Documentation is provided for the CryptExportKey function. This function is available from version 1.30 of the CSP.

I.5 Changes Forecast

The addition of encryption capabilities will be added at a later date whenever the government decides to allow the inclusion of key material allowing encryption/decryption.

I.5.1

II Purpose

The purpose of this document is to give any application programmer the necessary information to develop applications in where the Belgian electronic identity card is used. This document is limited to programmer's information regarding the Belgian card middleware.

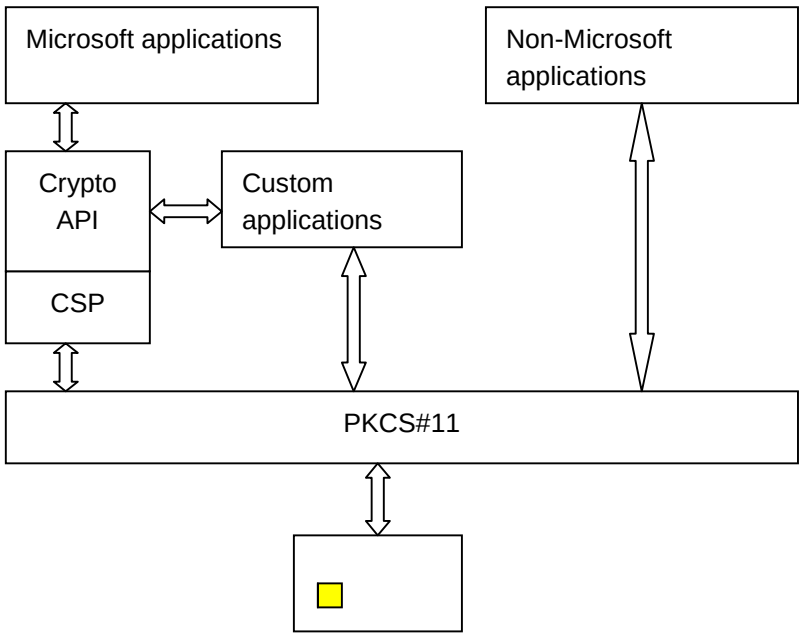
This document is preliminary and is subject to change without prior notice.

III Table of Contents

IV Programmers Guide

IV.1 Introduction

The Belgian identity card middleware is software that is placed between the application implementing security features (digital signatures) and the device actually performing the cryptographic operations (the smartcard).
The middleware itself consists out of two independent interface implementations (see figure below). Although the implementations are independent, one makes use of the other. For the Microsoft® standard applications (Office, Outlook…) a Cryptographic Service Provider (CSP) is created that implements the cryptographic operations from the smartcard. An application will never call this implementation directly but through a standard interface called Crypto API. The CSP implementation makes use of the second implemented interface, PKCS#11. This interface is used by non-Microsoft standard applications.
When a new application is created, it is up to the developer to decide which of the two interfaces will be used to offer cryptographic functionality to the user.



This document outlines both programming interfaces and how an application builder can make use of them.

IV.2 Assumptions

It is assumed that the user of this document has a working knowledge of cryptographic operations like digital signatures, hashing operations, key material …

Interfaces

As described in the previous section, there are two interfaces implemented in the middleware software: one interface that can be called directly (the PKCS#11 interface) and one interface that is called indirectly (the CSP).
This document outlines the usage of the interface and when appropriate where the implementation deviates from what is described in the standard documents. Although the parameters passed into the API and possible return values may be extended in some cases, the interface description (i.e. function prototypes) has not been modified in any way.

IV.3 The Crypto API interface

The Microsoft® Cryptographic API 2.0 (CryptoAPI) enables application developers to add authentication, encoding, and encryption to their Win32®-based applications. Application developers can use functions in the CryptoAPI without knowing anything about the underlying implementation, in much the same way as they can use a graphics library without knowing anything about the particular graphics hardware configuration.
The CSP part of the middleware establishes the link between the abstract CryptoAPI and the underlying PKCS#11 interface. The developer will never call any of the functions of the CSP directly but through the CryptoAPI. In the sections below a description will be given of the API calls that CryptoAPI reverts to the CSP for further processing. This document does not provide any detailed information on the operation of each API call. For this type of information please refer to the Microsoft Developer Network.
The Belgian identity card only supports digital signature operations. All functions not related to this cryptographic operation are not implemented. When at a later date the Belgian government would decide to allow the user of the electronic identity card to add key material on the card that supports encryption then the CSP will be extended to allow for this additional functionality. Furthermore the Belgian identity card contains two key pairs that can be used for digital signatures (both authentication and non-repudiation). Because of this issue some of the parameters passed to the Crypto API functions have no meaning. For example in the call CryptGetUserKey a parameter called dwKeySpec is passed. This parameter is used to define which type of key to get, an AT_KEYEXCHANGE key or an AT_SIGNATURE key. However, in the case of the Belgium Identity Card CSP this parameter does not suffice to determine which signature key to load. In this case the container that contains the correct certificate must be passed to CryptAcquireContext a futher call to CryptGetUserKey will then be completed successfully.
Although the CSP only supports digital signatures, it is still registered as a PROV_RSA_FULL type of CSP. This is done in order to allow the usage of the CSP in standard Microsoft® applications. Calling Crypto API functions that are not used in a digital signature context will result in a returned error value indicating that the API function is not implemented.
The description in this document applies to version 1.20 of the CSP.

IV.3.1 CryptAcquireContext

```
BOOL WINAPI CryptAcquireContext(HCRYPTPROV *phProv,
                                LPCTSTR pszContainer,
                                LPCTSTR pszProvider,
                                DWORD dwProvType,
                                DWORD dwFlags);
```

The parameter *pszContainer* contains the name of the key container that contains a specific key on the identity card. The names of the containers on the identity card can be obtained through a call to **CryptGetProvParam**.

The parameter *dwFlags* can be set to the following values (according MSDN):

0 (equivalent to CRYPT_SCKEYSET)
CRYPT_VERIFYCONTEXT
CRYPT_NEWKEYSET
CRYPT_MACHINE_KEYSET
CRYPT_DELETEKEYSET

As the key material of the Belgian identity card are stored on a smartcard and the user has no permissions to create new key sets on the card the parameter values CRYPT_NEWKEYSET, CRYPT_MACHINE_KEYSET and CRYPT_DELETEKEYSET are not supported. Using these values will generate the error NTE_BAD_FLAGS.
An extra value for this parameter is defined, CRYPT_SCKEYSET. With this value the developer defines that a context for the key as defined in the *pszContainer* parameter is acquired.
For hashing operations only, a base CSP is used. If for some reason loading the base CSP would fail, then following error code will be set through **SetLastError()**:

ERR_CANNOT_LOAD_BASE_CSP (0x1000)

IV.3.2 CryptReleaseContext

```
BOOL WINAPI CryptReleaseContext(HCRYPTPROV hProv,
                                DWORD dwFlags);
```

This API call is implemented as defined by MSDN.

IV.3.3 CryptGenerateKey

```
BOOL WINAPI CryptGenKey( HCRYPTPROV hProv,
                         ALG_ID Algid,
                         DWORD dwFlags,
                         HCRYPTKEY *phKey);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.4 CryptDeriveKey

```
BOOL WINAPI CryptDeriveKey(HCRYPTPROV hProv,
                           ALG_ID Algid,
                           HCRYPTHASH hBaseData,
                           DWORD dwFlags,
                           HCRYPTKEY *phKey);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.5 CryptDestroyKey

```
BOOL WINAPI CryptDestroyKey(HCRYPTKEY hKey);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.6 CryptSetKeyParam

```
BOOL WINAPI CryptSetKeyParam(HCRYPTKEY hKey,
                             DWORD dwParam,
                             BYTE *pbData,
                             DWORD dwFlags);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.7 CryptGetKeyParam

```
BOOL WINAPI CryptGetKeyParam(HCRYPTKEY hKey,
                             DWORD dwParam,
                             BYTE *pbData,
                             DWORD *pcbData,
                             DWORD dwFlags);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.8 CryptSetProvParam

```
BOOL WINAPI CryptSetProvParam(HCRYPTPROV hProv,
                              DWORD dwParam,
                              BYTE *pbData,
                              DWORD dwFlags);
```

According to the MSDN documentation the *dwParam* parameter can be set to the following values:

PP_CLIENT_HWND
PP_KEYSET_SEC_DESCR

The latter parameter does not make any sense since the key material in case of the Belgian identity card is stored in the smartcard instead of in the registry. Therefore this parameter will be ignored.

IV.3.9 CryptGetProvParam

```
BOOL WINAPI CryptGetProvParam(HCRYPTPROV hProv,
                              DWORD dwParam,
                              BYTE *pbData,
                              DWORD *pcbData,
                              DWORD dwFlags);
```

This API call is implemented as described in the MSDN documentation with the exception of the PP_KEYSET_SEC_DESCR parameter which is ignored.
For the PP_IMPTYPE parameter the value CRYPT_IMPL_MIXED is returned because the signing operation is handled by hardware (i.e. the smartcard) while the hashing operation is handled by the base cryptographic provider.

IV.3.10 CryptSetHashParam

```
BOOL WINAPI CryptSetHashParam(HCRYPTHASH hHash,
                              DWORD dwParam,
                              BYTE *pbData,
                              DWORD dwFlags);
```

This API call is implemented as described in the MSDN documentation.
The parameter *dwParam* = HP_HASHVAL is implemented but should be used with caution. This parameter was defined to give applications the ability to sign hash values, without having access to the base data. Because the application (much less the user) can have no idea what is being signed, this operation is intrinsically risky.

IV.3.11 CryptGetHashParam

```
BOOL WINAPI CryptGetHashParam(HCRYPTHASH hHash,
                              DWORD dwParam,
                              BYTE *pbData,
                              DWORD *pcbData,
                              DWORD dwFlags);
```

This API call is implemented as described in the MSDN documentation.

IV.3.12 CryptExportKey

```
BOOL WINAPI CryptExportKey(HCRYPTKEY hKey,
                           HCRYPTKEY hExpKey,
                           DWORD dwBlobType,
                           DWORD dwFlags,
                           BYTE *pbData,
                           DWORD *pcbDataLen);
```

This function can be used to export the public key associated with the hKey parameter. A handle to a public key can be obtained through a call to CryptGetUserKey. Since the private keys are stored on a smartcard and exporting of private keys is not permitted only PUBLICKEYBLOB can be defined as dwBlobType. Because only public keys can be exported, the parameter hExpKey is not used and should therefore be set to NULL. The public key is returned pbData parameter. To obtain the length of the data to will be returned the parameter pbData can be set to NULL. The length of the data that will be returned is then placed in pcbDataLen. If the buffer passed to this function is not large enough, the error ERROR_MORE_DATA will be returned and the correct buffer length is put in the pcbDataLen parameter.

IV.3.13 CryptImportKey

```
BOOL WINAPI CryptImportKey(HCRYPTPROV hProv,
                           BYTE *pbData,
                           DWORD dwDataLen,
                           HCRYPTKEY hPubKey,
                           DWORD dwFlags,
                           HCRYPTKEY *phKey);
```

Since the key material on the Belgian identity card is pre-installed by the government and the user does not have the permissions to create additional key pairs, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.14 CryptEncrypt

```
BOOL WINAPI CryptEncrypt(HCRYPTKEY hKey,
                         HCRYPTHASH hHash,
                         BOOL Final,
                         DWORD dwFlags,
                         BYTE *pbData,
                         DWORD *pcbData,
                         DWORD cbBuffer);
```

Currently the key usages as they are defined by the Belgian government do not support encryption. Therefore this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.
If at some future date key material supporting encryption can be added to the electronic ID card then this function will be implemented as well.

IV.3.15 CryptDecrypt

```
BOOL WINAPI CryptDecrypt(HCRYPTKEY hKey,
                         HCRYPTHASH hHash,
                         BOOL Final,
                         DWORD dwFlags,
                         BYTE *pbData,
                         DWORD *pcbData);
```

Currently the key usages as they are defined by the Belgian government do not support encryption. Therefore this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

If at some future date key material supporting encryption can be added to the electronic ID card then this function will be implemented as well.

IV.3.16 CryptCreateHash

```
BOOL WINAPI CryptCreateHash(HCRYPTPROV hProv,
                            ALG_ID Algid,
                            HCRYPTKEY hKey,
                            DWORD dwFlags,
                            HCRYPTHASH *phHash);
```

This API call is implemented as it is described in the MSDN documentation. One additional error can be returned through **SetLastError ()**:

ERR_INVALID_PROVIDER_HANDLE (0x1001)

This error indicates that the handle as it is specified by *hProv* could not be found (i.e. it was not created using **CryptAcquireContext ()**)

The actual processing of this call is delegated to a base CSP.

IV.3.17 CryptHashData

```
BOOL WINAPI CryptHashData(HCRYPTHASH hHash,
                          BYTE *pbData,
                          DWORD cbData,
                          DWORD dwFlags);
```

This API call is implemented as it is described in the MSDN documentation. In the parameter *dwFlags* one value (apart from 0) can be specified: CRYPT_USERDATA. Depending on which base CSP is chosen it may or may not be implemented. For instance the Microsoft Base CSP does not implement this parameter.

The actual processing of this call is delegated to a base CSP.

IV.3.18 CryptHashSessionKey

```
BOOL WINAPI CryptHashSessionKey(HCRYPTHASH hHash,
                                HCRYPTKEY hKey,
                                DWORD dwFlags);
```

Since some of the underlying calls necessary to use this function are currently not implemented by this CSP also this call is not available. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.3.19 CryptSignHash

```
BOOL WINAPI CryptSignHash(HCRYPTHASH hHash,
                          DWORD dwKeySpec,
                          LPCTSTR sDescription,
                          DWORD dwFlags,
                          BYTE *pbSignature,
                          DWORD *pdwSigLen);
```

This API call is implemented as it is defined in the MSDN documentation. When this function is called, an attempt is made to connect and log on to the Belgian identity card (smartcard). If any of these operations fail, following error can be generated through **SetLastError ()**:

ERR_CANNOT_LOGON_TO_TOKEN (0x1004)

In order to sign the given hash data, some information (e.g. key length) needs to be read from the smartcard. If an error occurs during this operation, following error will be generated through **SetLastError ()**:

ERR_CANNOT_GET_TOKEN_SLOT_INFO (0x1003)

The signing mechanism used to produce digital signatures is CKM_RSA_PKCS. Please refer to the PKCS#11 documentation for more detailed information on this mechanism.

Currently following hashing algorithms can be used to sign data : MD2, MD4, MD5, SHA-1 and SSL3 SHAMD5. Although the MDx hashing algorithms are still available for backward compatibility it is suggested that new applications use SHA-1.

IV.3.20 CryptDestroyHash

```
BOOL WINAPI CryptDestroyHash(HCRYPTHASH hHash);
```

This API call is implemented as it is defined in the MSDN documentation.

IV.3.21 CryptVerifySignature

```
BOOL WINAPI CryptVerifySignature(HCRYPTHASH hHash,
                                 BYTE *pbSignature,
                                 DWORD dwSigLen,
                                 HCRYPTKEY hPubKey,
                                 LPCTSTR sDescription,
                                 DWORD dwFlags);
```

This function is implemented for convenience reasons. This call is delegated to the base CSP.

IV.3.22 CryptGenRandom

```
BOOL WINAPI CryptGenRandom(HCRYPTPROV hProv,
                           DWORD dwLen,
                           BYTE *pbBuffer);
```

This API call is implemented as it is defined in the MSDN documentation. The data entered through pbBuffer will be used as seed for the random generation.

IV.3.23 CryptGetUserKey

```
BOOL CryptGetUserKey(HCRYPTPROV hProv,
                     DWORD dwKeySpec,
                     HCRYPTKEY *phUserKey);
```

This call returns a handle to the public key of the key container that was defined through CryptAcquireContext. Specifying AT_SIGNATURE for the parameter dwKeySpec is not enough because with that information the CSP can still not determine which signature key to return. Therefore the key to load must first be specified through CryptAcquireContext.

IV.3.24 CryptDuplicateHash

```
BOOL WINAPI CryptDuplicateHash(HCRYPTHASH hHash,
                               DWORD *pdwReserved,
                               DWORD dwFlags,
                               HCRYPTHASH phHash);
```

This API call is implemented as it is defined in the MSDN documentation.

IV.3.25 CryptDuplicateKey

```
BOOL WINAPI CryptDuplicateKey(HCRYPTKEY hKey,
                              DWORD *pdwReserved,
                              DWORD dwFlags,
                              HCRYPTKEY* phKey);
```

Since the key material is stored on a smartcard and cannot be retrieved this function does not make sense. Therefore, this API call is not implemented. Calling this function anyway, will generate the error **E_NOTIMPL** set through **SetLastError ()**.

IV.4 The PKCS#11 interface

The PKCS#11 (v2.11) interface is used by non-Microsoft applications like for instance Netscape. Also custom application can make use of this interface instead of the CryptoAPI interface. The PKCS#11 interface is sometimes also called Cryptoki.
A detailed description of this interface can be found on the website of RSA Laboratories (http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/).

IV.4.1 API calls implemented

IV.4.1.1 General Purpose functions

C_Initialize,
C_Finalize
C_GetInfo
C_GetFunctionList

IV.4.1.2 Slot and token management functions

C_GetSlotList
C_GetSlotInfo
C_GetTokenInfo
C_GetMechanismList
C_GetMechanismInfo
C_WaitForSlotEvent
C_SetPin

IV.4.1.3 Session management functions

C_OpenSession
C_CloseSession
C_CloseAllSessions
C_GetSessionInfo
C_Login
C_Logout

IV.4.1.4 Object Management functions

C_FindObjectsInit
C_FindObjects
C_FindObjectsFinal
C_GetAttributeValue

IV.4.1.5 Signing Functions

C_SignInit

C_Sign
C_SignUpdate
C_SignFinal

IV.4.1.6 Digest Functions

C_DigestInit
C_Digest
C_DigestUpdate
C_DigestFinal

IV.4.1.7 Random Generation Functions (to be confirmed soon)

C_SeedRandom
C_GenerateRandom

IV.4.2 Supported Signature mechanisms

For signatures:
- CKM_RSA_X_509

- CKM_RSA_PKCS: both ASN.1-wrapped and pure hashes (MD5, SHA1, SHA1+MD5, RIPEMD160, in the case 20 bytes are given, a SHA-1 hash is assumed)

- CKM_RIPEMD160_RSA_PKCS, CKM_SHA1_RSA_PKCS, CKM_MD5_RSA_PKCS

- When supported by the electronic ID card following signature mechanisms will also b supported by the middleware: CKM_RSA_PKCS_PSS, CKM_SHA1_RSA_PKCS_PSS

For digests:
CKM_SHA_1, CKM_RIPEMD160, CKM_MD5

IV.4.3 Slot and token information

There will be a virtual slot/token for each PIN (so in the case of the Belgian electronic identity card this means 1 slot/token).
The public keys, private keys and certificates that belong together will have the same CKA_ID object attribute.

IV.4.4 Behavior in case of a PIN pad reader

In this case, CK_TOKEN_INFO will have the CKF_PROTECTED_AUTHENTICATION_PATH flag set and the application should give a NULL PIN with C_Login. When a C_Login is called, the PKCS#11 library will present the user a dialog asking to enter the PIN on the PIN pad for placing a signature or identification.

IV.4.5 Behaviour with the non-repudiation key

If a signature is requested with this key, the PKCS#11 library itself will show a GUI to either ask the user to enter her PIN, or to ask the user to supply her PIN at the PIN pad reader.