Projecto Cartão de Cidadão

Manual técnico do Middleware Cartão de Cidadão

Julho 2007

Versão 1.0

Este documento apresenta a descrição das interfaces disponibilizadas pelo "middleware"do Cartão de Cidadão. Destina-se a técnicos, analistas e arquitectos de tecnologias de informação com o objectivo de promover o desenvolvimento de aplicações que interajam com o novo documento de identificação nacional.

Tabela de Conteúdos

1. Visão Global

As seguintes interfaces podem ser encontradas no middleware do Cartão de Cidadão:

- CryptoAPI/CSP
- PKCS#11
- eID lib (= a 'SDK' ou 'Software Development Kit')

A CSP está apenas disponível em ambiente Windows; as outras 2 bibliotecas estão disponíveis em Windows, Linux e Mac OS X. As primeiras 2 interfaces são APIs standard para operações criptográficas; a eID lib tem uma API non-standard que é direccionada à funcionalidade não-criptográfica do Cartão de Cidadão.

No Windows, estas bibliotecas podem ser encontradas na pasta System32; no Mac OS X no directório /usr/local/lib e em Linux depende do directório de instalação.

Este documento fornece informação para programadores sobre como desenvolver aplicações com base nestas interfaces.

Para informações ou guias sobre como usar estas bibliotecas, por favor consulte os Manuais de Utilizador respectivos.

Adicionalmente, as seguintes aplicações estão presentes no middleware:

eID GUI

Esta aplicação pode ser usada para ver e gerir a informação no cartão eID:

- Ler e mostrar informação sobre o cidadão e fotografia
- Ler e mostrar a morada do cidadão
- Ler os certificados do governo e do cidadão
- Registar os certificados do governo e do cidadão (apenas Windows)
- Gestão de códigos PIN (Testar o PIN, Alterar o PIN)
- Gerir a Pasta pessoal
- Ver informação específica de ministérios

Tray applet

Esta aplicação é instalada como uma funcionalidade da área de notificação. No Windows, aparece normalmente no canto inferior direito do ecrã, no Mac no canto superior direito do ecrã. Quando activada (o utilizador pode

desactivá-la), verifica se um cartão eID está inserido. Após inserir o cartão será mostrada a fotografia do cidadão durante uns segundos.

Irá também registar automaticamente (se esta opção estiver activada) os certificados do cartão na Microsoft certificate store, caso ainda não estejam registados. Quando o cartão é removido os certificados registados são automaticamente removidos da certificate store (se esta opção estiver activada).
Este modo é também conhecido como modo "Kiosk". Esta funcionalidade a nível de certificados é apenas implementada na plataforma Windows devido às outras plataformas (Mac e Linux) não suportarem o conceito de "certificate stores".

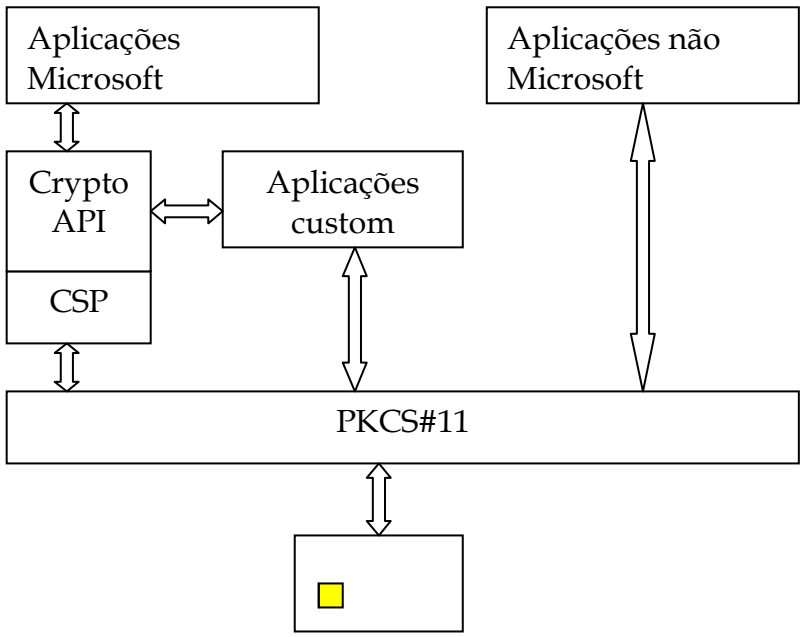
Por favor consulte o Manual de Utilizador para mais informação sobre estas aplicações.

2. Documentação CSP

2.1. Introdução

Para as aplicações standard Microsoft® (Office, Outlook...) é criado um Cryptographic Service Provider (CSP) que implementa as operações criptográficas do smartcard. Uma aplicação nunca chamará esta implementação directamente mas sim através de uma interface standard chamada Crypto API. A implementação CSP utiliza a segunda interface implementada, PKCS#11. Esta interface é usada por aplicações não standard Microsoft.

Quando uma nova aplicação é criada, é o programador que decide qual das



duas interfaces utilizar para oferecer funcionalidade criptográfica ao utilizador.

2.2. A interface Crypto API

O Microsoft® Cryptographic API 2.0 (CryptoAPI) permite a programadores de aplicações adicionarem funcionalidade de autenticação, encoding, e encriptação às suas aplicações baseadas em Win32®. Os programadores de aplicações podem usar funções da CryptoAPI sem necessidade de

conhecerem a implementação subjacente, da mesma forma que podem usar uma biblioteca gráfica sem terem conhecimento sobre a configuração de hardware.

A componente CSP do middleware estabelece a ligação entre a abstracta CryptoAPI e a interface PKCS#11 subjacente. O programador nunca irá chamar nenhuma das funções do CSP directamente mas sim através da CryptoAPI. Nas secções abaixo será apresentada uma descrição das chamadas à API que a CryptoAPI encaminha para o CSP para processamento. Este documento não fornece informação detalhada sobre a operação de cada chamada à API. Para este tipo de informação por favor consulte a Microsoft Developer Network (MSDN).

O cartão de identidade Português apenas suporta operações de assinatura digital. Todas as funções não relacionadas com esta operação criptográfica não são implementadas. O cartão contém dois pares de chaves que podem ser usados para assinaturas digitais; o primeiro para autenticação e o segundo para não-repudio (assinaturas legalmente vinculativas). Devido a isto alguns parâmetros passados para as funções da CryptoAPI não têm significado. Por exemplo na chamada à API CryptGetUserKey é passado um parâmetro chamado dwKeySpec. Este parâmetro é usado para definir qual o tipo de chave a obter, AT_KEYEXCHANGE ou AT_SIGNATURE. No entanto, no caso do CSP do Cartão de Cidadão este parâmetro não é suficiente para determinar qual a chave de assinatura a carregar. Neste caso o contentor que contém o certificado correcto deve ser passado para CryptAcquireContext e então a chamada para a função CryptGetUserKey será completada com sucesso.

Apesar do CSP apenas suportar assinaturas digitais, está mesmo assim registado como um CSP de tipo PROV_RSA_FULL. Isto é feito de forma a permitir a utilização do CSP em aplicações standard Microsoft®. Chamar funções da Crypto API que não são usadas num contexto de assinatura digital resultará num erro indicando que a funcionalidade não está implementada.

2.2.1. CryptAcquireContext

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

O parâmetro *pszContainer* contém o nome do key container que tem uma determinada chave do cartão. Os nomes dos containers existentes no cartão podem ser obtidos através de uma chamada à função **CryptGetProvParam**.

O parâmetro *dwFlags* pode ser definido para os seguintes valores (de acordo com a MSDN):

0 (equivalente a CRYPT_SCKEYSET)

CRYPT_VERIFYCONTEXT

CRYPT_NEWKEYSET

CRYPT_MACHINE_KEYSET

CRYPT_DELETEKEYSET

Como a informação relativa a chaves do Cartão de Cidadão está guardada num smartcard e o utilizador não tem permissões para criar novos conjuntos de chaves, os valores CRYPT_NEWKEYSET, CRYPT_MACHINE_KEYSET e CRYPT_DELETEKEYSET não são suportados. A utilização destes valores irá gerar o erro NTE_BAD_FLAGS.

Está definido para este parâmetro um valor extra, CRYPT_SCKEYSET. Com este valor o programador define que é adquirido um contexto para a chave, definido no parâmetro *pszContainer*.

É utilizado base CSP, somente para operações de hashing. Se por algum motivo o carregamento do base CSP falhar, então o seguinte código de erro será definido através de **SetLastError()**:

ERR_CANNOT_LOAD_BASE_CSP (0x1000)

2.2.2. CryptReleaseContext

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

Esta chamada à API é implementada tal como definido pela MSDN.

2.2.3. CryptGenerateKey

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

Visto que as chaves e certificados do Cartão de Cidadão são pré-instalados pelo governo e o utilizador não tem permissões para criar novos pares de chaves, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através do **SetLastError ()**.

2.2.4. CryptDeriveKey

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

Visto que esta funcionalidade refere-se apenas a chaves de encryptação e estas não estão presentes no cartão, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através do **SetLastError ()**.

2.2.5. CryptDestroyKey

BOOL WINAPI CryptDestroyKey(HCRYPTKEY *hKey*);

Visto que as chaves e certificados do Cartão de Cidadão são pré-instalados pelo governo e o utilizador não tem permissões para criar novos pares de chaves, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através do **SetLastError ()**.

2.2.6. CryptSetKeyParam

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

Visto que as chaves e certificados do Cartão de Cidadão são pré-instalados pelo governo e o utilizador não tem permissões para criar novos pares de chaves, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através do **SetLastError ()**.

2.2.7. CryptGetKeyParam

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

Since the key material on the Portuguese 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 ()**.

2.2.8. CryptSetProvParam

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

De acordo com a documentação da MSDN o parâmetro *dwParam* pode ser definido para os seguintes valores:

PP_CLIENT_HWND

PP_KEYSET_SEC_DESCR

O último parâmetro não faz sentido visto que não é possível escrever informação sobre as chaves no cartão. Este parâmetro deverá ser ignorado.

2.2.9. CryptGetProvParam

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

Esta chamada à API é implementada com base na documentação MSDN à excepção do parâmetro PP_KEYSET_SEC_DESCR, que é ignorado.

Para o parâmetro PP_IMPTYPE é devolvido o valor CRYPT_IMPL_MIXED porque a operação de assinatura é executada pelo hardware (smartcard) enquanto que a operação de hashing é executada pelo base cryptographic provider.

2.2.10. CryptSetHashParam

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

Esta chamada à API é implementada com base na documentação MSDN.

O parâmetro *dwParam* = HP_HASHVAL é implementado mas deve ser usado com cuidado. Este parâmetro foi definido de forma a dar às aplicações a possibilidade de assinar hash values, sem ter acesso à base data. Porque a aplicação (e muito menos o utilizador) não pode ter ideia do que está a ser assinado, esta operação é intrinsecamente arrisacada.

2.2.11. CryptGetHashParam

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

Esta chamada à API é implementada com base na documentação MSDN.

2.2.12. CryptExportKey

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

Esta função pode ser usada para exportar a chave pública associada ao parâmetro hKey. O handle de uma chave pública pode ser obtido através de uma chamada à função CryptGetUserKey. Visto que as chaves privadas estão guardadas num smartcard e a exportação destas não é permitida, apenas PUBLICKEYBLOB pode ser definido como dwBlobType. Devido ao facto de apenas as chaves públicas poderem ser exportadas, o parâmetro hExpKey não é utilizado e deve portanto ser definido como NULL. A chave pública é retornada como parâmetro pbData. Para obter o comprimento dos dados o parâmetro pbData deve ser definido como NULL. O comprimento dos dados que será devolvido é então colocado no parâmetro pcbDataLen. Se o buffer para esta função não for suficientemente grande, será devolvido o erro ERROR_MORE_DATA, e o valor correcto para o comprimento do buffer será colocado no parâmetro pcbDataLen.

2.2.13. CryptImportKey

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

Visto que os chaves e certificados do Cartão de Cidadão são pré-instalados pelo governo e o utilizador não tem permissões para criar pares de chaves adicionais, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através de **SetLastError ()**.

2.2.14. CryptEncrypt

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

Tal como estipulado pelo Governo Português, não é suportada a utilização das chaves para encriptação. Deste modo esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através de **SetLastError ()**.

Caso no futuro sejam adicionadas chaves de encriptação ao Cartão de Cidadão, então esta função será também implementada.

2.2.15. CryptDecrypt

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

Tal como estipulado pelo Governo Português, não é suportada a utilização das chaves para encriptação. Deste modo esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através de **SetLastError ()**.

Caso no futuro sejam adicionadas chaves de encriptação ao Cartão de Cidadão, então esta função será também implementada.

2.2.16. CryptCreateHash

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

Esta chamada à API é implementada com base na documentação MSDN. Um erro adicional pode ser devolvido através de **SetLastError ()**:

ERR_INVALID_PROVIDER_HANDLE (0x1001)

Este erro indica que o handle esperado por *hProv* não foi encontrado (não foi criado usando **CryptAcquireContext ()**)

O processamento desta chamada é delegado a uma base CSP.

2.2.17. CryptHashData

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

Esta chamada à API é implementada com base na documentação MSDN. No parâmetro *dwFlags* um valor (excepto 0) pode ser especificado: CRYPT_USERDATA. Dependendo da base CSP escolhida poderá ou não ser implementado. Por exemplo a Microsoft Base CSP não implementa este parâmetro.

O processamento desta chamada é delegado a uma base CSP.

2.2.18. CryptHashSessionKey

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

Visto que algumas das chamadas subjacentes necessárias para usar esta função não são de momento implementadas por este CSP, esta chamada também não está disponível. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através de **SetLastError ()**.

2.2.19. CryptSignHash

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

Esta chamada à API é implementada com base na documentação MSDN. Quando esta função é chamada, é efectuada uma tentativa de conexão ao Cartão de Cidadão (smartcard). Se alguma destas operações falhar, o seguinte erro pode ser gerado através de **SetLastError ()**:

ERR_CANNOT_LOGON_TO_TOKEN (0x1004)

De modo a assinar os dados hash, é necessário ler alguma informação (por exemplo o comprimento da chave) do smartcard. Caso ocorra um erro durante esta operação a seguinte mensagem de erro será gerada através de **SetLastError ()**:

ERR_CANNOT_GET_TOKEN_SLOT_INFO (0x1003)

O mecanismo de assinatura utilizado para produzir assinaturas digitais é CKM_RSA_PKCS. Por favor consulte a documentação PKCS#11 para informação detalhada sobre este mecanismo.

Os seguintes algoritmos de hashing podem ser usados para assinatura de dados: MD2, MD4, MD5, SHA-1 e SSL3 SHAMD5. Apesar dos algoritmos de hashing MDx ainda estarem disponíveis para retrocompatibilidade, é aconselhado o uso de SHA-1 para novas aplicações.

2.2.20. CryptDestroyHash

BOOL WINAPI CryptDestroyHash(HCRYPTHASH *hHash*);

Esta chamada à API é implementada com base na documentação MSDN.

2.2.21. CryptVerifySignature

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

Esta função é implementada por motivos de conveniência. Esta chamada é delegada para a base CSP.

2.2.22. CryptGenRandom

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

Esta chamada à API é implementada com base na documentação MSDN. Os dados inseridos através de pbBuffer serão usados como origem para a geração aleatória.

2.2.23. CryptGetUserKey

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

Esta chamada devolve um handle para a chave pública do contentor de chaves que foi definido através de CryptAcquireContext. Especificar AT_SIGNATURE para o parâmetro dwKeySpec não é suficiente porque com essa informação o CSP não consegue determinar que chave de assinatura devolver. Por este motivo a chave a carregar tem de ser primeiro especificada através de CryptAcquireContext.

2.2.24. CryptDuplicateHash

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

Esta chamada à API é implementada com base na documentação MSDN.

2.2.25. CryptDuplicateKey

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

Visto que as chaves e certificados são guardados no smartcard, esta chamada à API não é implementada. Chamar esta função irá gerar o erro **E_NOTIMPL** definido através de **SetLastError ()**.

3. Documentação PKCS#11

3.1. A interface PKCS#11

A interface PKCS#11 (v2.20) é utilizada por aplicações não Microsoft como por exemplo o Netscape. Aplicações desenvolvidas podem recorrer a este interface em vez do interface CryptoAPI. A interface PKCS#11 é por vezes chamada de Cryptoki.

Uma descrição detalhada desta interface está disponível no website da RSA Laboratories (http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/).

3.1.1. Chamadas à API implementadas

3.1.1.1. Funções Gerais

C_Initialize,

C_Finalize

C_GetInfo

C_GetFunctionList

3.1.1.2. Funções de gestão de Slot e token

C_GetSlotList

C_GetSlotInfo

C_GetTokenInfo

C_GetMechanismList

C_GetMechanismInfo

C_WaitForSlotEvent (only non-blocking)

C_SetPin

3.1.1.3. Funções de gestão de sessão

C_OpenSession

C_CloseSession

C_CloseAllSessions

C_GetSessionInfo

C_Login

C_Logout

3.1.1.4. Funções de gestão de objectos

C_FindObjectsInit

C_FindObjects

C_FindObjectsFinal

C_GetAttributeValue

3.1.1.5. Funções de assinatura

C_SignInit

C_Sign

C_SignUpdate

C_SignFinal

3.1.1.6. Funções de digest

C_DigestInit

C_Digest

C_DigestUpdate

C_DigestFinal

3.1.1.7. Funções de geração aleatória (a aguardar confirmação)

C_SeedRandom

C_GenerateRandom

3.1.2. Mecanismos de assinatura suportados

Para assinaturas:

- CKM_RSA_PKCS: both ASN.1-wrapped e hashes puros (MD5, SHA1, SHA1+MD5, RIPEMD160)
- CKM_RIPEMD160_RSA_PKCS, CKM_SHA1_RSA_PKCS, CKM_MD5_RSA_PKCS

Para digests:

- CKM_SHA_1, CKM_RIPEMD160, CKM_MD5

3.1.3. Informações de Slot e token

O cartão será representado como um token PKCS#11 com o PIN de Autenticação do Cidadão servindo como User PIN; não está presente nenhum SO PIN.

O PIN de Assinatura do Cidadão estará ocultado; caso seja necessário este será requisitado através de uma caixa de diálogo.

As chaves públicas, chaves privadas e os certificados que façam parte do mesmo conjunto terão o mesmo atributo no objecto CKA_ID.

3.1.4. Comportamento da chave de não-repúdio

Se uma assinatura for solicitada com esta chave, a própria biblioteca PKCS#11 mostrará uma caixa de diálogo pedindo ao utilizador para inserir o PIN.

4. Documentação eID Lib API

A SDK está destinada a organizações cujo objectivo seja desenvolver aplicações que utilizam o Cartão de Cidadão. Este kit de desenvolvimento lida apenas com os dados identificativos do cidadão e não com operações criptográficas.

O kit de desenvolvimento é fornecido como:

- Uma interface C/C++, como biblioteca dinâmica
- Uma biblioteca Java wrapper (JNI) sobre uma interface C/C++
- Uma biblioteca C# wrapper para .NET sobre uma interface C/C++

4.1. Gestão de versões e compatibilidade

O Toolkit gere automaticamente todas as diferentes versões dos cartões. Ao trabalhar com o Toolkit não há necessidade de preocupação com a forma como os dados estão gravados no cartão, visto estes estarem disponíveis de maneira uniforme através da API.

Existem funções de baixo nível para obter as várias versões dos componentes do cartão, mas isto é apenas direccionado a programadores que necessitam de aceder a características muito específicas do cartão – uma aplicação comum não deverá ter que se preocupar com a versão do cartão.

4.2. Inserção de PIN

Várias funções aceitam um parâmetro de inserção de referência PIN. Caso uma referência PIN seja fornecida e a função obtiver um "access denied" quando tenta aceder a um recurso no cartão, esta irá automaticamente pedir ao utilizador para fornecer um PIN e tentará aceder novamente ao recurso (caso a verificação do PIN tenha sido bem sucedida). Isto é uma verificação "just-in-time" do PIN, visto que este só será solicitado quando necessário. Por exemplo, um PIN permanente pode ter sido inserido previamente e ainda ser válido. Neste caso, não será solicitado novamente.

4.3. Aplicação Multi-threaded

A biblioteca não é "thread-safe". É da responsabilidade da aplicação que a chama de não usar a biblioteca simultaneamente em threads paralelas. Nota: O CSP é "thread-safe", mas não poderá chamar o CSP numa thread e o Toolkit noutra thread.

4.4. Organização API

As funções estão divididas em 4 categorias:

- *Funções* ***Initialisation*** *e* ***termination***, mandatórias para iniciar e terminar a utilização do toolkit
- *Funções de* ***Identity***, usadas para obter dados identificativos (nome, morada, etc.) do cartão
- *Funções* ***General purpose de alto nível***, usadas para aceder a dados de uma forma genérica (ficheiros, PIN), principalmente em outras aplicações que não as de *Identity*. Não há necessidade de usar estas funções para aceder a dados identificativos.

4.5. Funções de Initialisation e termination

Estas funções são necessárias para inicializar e terminar a eID Lib.

Funções:

- PTEID_Init()
- PTEID_Exit()

4.6. Funções de Identity

Todas as funções de *Identity* são auto-suficientes. Isto é, não é necessário chamar nenhuma outra função em conjunto com uma função *identity* (excepto as de inicialização e término). Não é necessário inserir um PIN para ler ficheiros de identity. Todas estas funções podem ser chamadas independentemente do estado presente do cartão – caso outra DF (Data File) que não a identity estiver seleccionada, etc.

Importante: certifique-se que leu a secção relativa às funções SOD abaixo!

Funções:

- PTEID_GetID()
- PTEID_GetAddr()
- PTEID_GetPic()
- PTEID_GetCertificates()
- PTEID_GetPINs()
- PTEID_GetTokenInfo()

4.7. Funções General purpose de alto nível

Estas funções dão acesso – integradas com o toolkit – a funções gerais para aplicações que necessitem de efectuar outras acções para além do acesso aos dados identificativos (nome, morada, etc.)

Funções

- PTEID_SelectADF()
- PTEID_ReadFile()
- PTEID_WriteFile()
- PTEID_VerifyPIN()
- PTEID_ChangePIN()
- PTEID_UnblockPIN()
- PTEID_UnblockPIN_Ext()
- PTEID_IsActivated()
- PTEID_Activate()

4.8. Funções relacionadas com CVC

Para assegurar que os dados no cartão são genuínos, um ficheiro SOD é colocado no cartão contendo as hashes criptográficas assinadas sobre os dados identificativos, a morada, a fotografia e a chave de autenticação pública do cartão (pelo menos um é usado para autenticação CVC).

As seguintes funções usarão este SOD para verificação:

- os dados de identificação: PTEID_GetID()
- a morada: PTEID_GetAddr() and PTEID_CVC_GetAddr()
- a fotografia: PTEID_GetPic()

- a chave de autenticação pública do cartão: PTEID_CVC_Init()

As seguintes funções são fornecidas para controlar a verificação SOD:

- PTEID_SetSODChecking ()
- PTEID_SetSODCAs()

4.9. Detalhes da API C/C++

A API C/C++ é descrita em detalhe abaixo:

Para a API C# sobre esta API C/C++, por favor consulte as header files no directório "dotnet" da SDK.

Para a API Java sobre esta API C/C++, por favor consulte os Javadocs no directório "java" da SDK.

```
PTEIDLIB_API long
PTEID_Activate
                    ( char *           pszPin,
                      unsigned char *  pucDate,
                      unsigned long    ulMode
                    )
```

Activar o cartão (= actualizar um ficheiro específico do cartão).

Caso o cartão já tenha sido activado, é devolvido o erro SC_ERROR_NOT_ALLOWED.

Parâmetros:

pszPin in: valor do PIN de Activação

pucDate in: a data corrente no formato DD MM YY YY em formato BCD (4 bytes), ex: {0x17 0x11 0x20 0x06} para 17 de Novembro de 2006)

ulMode in: modo: MODE_ACTIVATE_BLOCK_PIN para bloquear o PIN de Activação, ou 0 para o inverso (deve apenas ser usado para testes).

```
PTEIDLIB_API long
PTEID_ChangePIN
                    (unsigned char   PinId,
                     char *          pszOldPin,
                     char *          pszNewPin,
                     long *          triesLeft
                    )
```

Mudar um PIN.

Parâmetros:

PinId	in: o PIN ID, ver o registo PTEID_Pins
pszOldPin	in: o valor actual do PIN, caso seja NULL o PIN será solicitado ao utilizador
pszNewPin	in: o novo valor do PIN, caso seja NULL o PIN será solicitado ao utilizador
triesLeft	out: as restantes tentativas de PIN

```
PTEIDLIB_API long
PTEID_CVC_Authenticate
                    ( unsigned char *   pucSignedChallenge,
                      int               iSignedChallengeLen
                    )
```

Concluir a autenticação CVC com o cartão, para ser chamado após PTEID_CVC_Init()

Parâmetros:

pucSignedChallenge	in: no challenge que foi assinado pela chave privada correspondente ao CVC
iSignedChallengeLen	in: o comprimento de ucSignedChallenge tem de ser 128

```
PTEIDLIB_API long PTEID_CVC_GetAddr( PTEID_ADDR * AddrData )
```

Ler o ficheiro de morada sobre um "canal CVC" e colocar os conteúdos num registo PTEID_ADDR.

PTEID_CVC_Init() e PTEID_CVC_Authenticate() bem sucedidos terão de ser efectuados anteriormente.

Parâmetros:

AddrData	out: o endereço de um registo PTEID_ADDR

```
PTEIDLIB_API long
PTEID_CVC_Init          ( const unsigned char *   pucCert,
                          int                     iCertLen,
                          unsigned char *         pucChallenge,
                          int                     iChallengeLen
                        )
```

Iniciar uma autenticação CVC com o cartão.

O challenge resultante deverá ser assinado com a chave privada correspondente ao certificado CVC (raw RSA signature) e fornecido na função PTEID_CVC_Authenticate().

Parâmetros:

pucCert	in: o CVC como um byte array
iCertLen	in: o comprimento de ucCert
pucChallenge	out: o challenge que será assinado pela chave CVC privada
iChallengeLen	in: o comprimento reservado para ucChallenge, deve ser 128

```
PTEIDLIB_API long
PTEID_CVC_ReadFile      ( unsigned char *   file,
                          int               filelen,
                          unsigned char *   out,
                          unsigned long *   outlen
                        )
```

Ler os conteúdos de um ficheiro sobre um "canal CVC".

PTEID_CVC_Init() e PTEID_CVC_Authenticate() bem sucedidos terão de ser efectuados anteriormente. Se *outlen for menor que os conteúdos do ficheiro, apenas os bytes *outlen serão lidos. Se *outlen for maior os conteúdos do ficheiro são devolvidos sem erro.

Parâmetros:

file	in: o caminho do ficheiro a ler (e.g. {0x3F, 0x00, 0x5F, 0x00, 0xEF, 0x05}
filelen	in: o coprimento do caminho do ficheiro (ex: 6)
out	out: o buffer para armazenar os conteúdos do ficheiro
outlen	out: o número de bytes a ler / número de bytes lidos.

PTEIDLIB_API long PTEID_CVC_WriteAddr(const PTEID_ADDR * *AddrData* **)**

Escrever o ficheiro de endereço sobre um "canal CVC". PTEID_CVC_Init() e PTEID_CVC_Authenticate() bem sucedidos terão de ser efectuados anteriormente.

Parâmetros:

AddrData in: o endereço de um registo PTEID_ADDR

```
PTEIDLIB_API long
PTEID_CVC_WriteFile
                        ( unsigned char *          file,
                          int                      filelen,
                          unsigned long            ulFileOffset,
                          const unsigned char *    in,
                          unsigned long            inlen,
                          unsigned long            ulMode
                        )
```

Escrever para um ficheiro no cartão sobre um "canal CVC"

PTEID_CVC_Init() e PTEID_CVC_Authenticate() bem sucedidos terão de ser efectuados anteriormente

Parâmetros:

file	in: o caminho do ficheiro a ler (ex: {0x3F, 0x00, 0x5F, 0x00, 0xEF, 0x05})
filelen	in: o comprimento do caminho do ficheiro (ex: 6)
ulFileOffset	in: escolher qual o offset no ficheiro por onde iniciar a escrita
in	in: os conteúdos do ficheiro
inlen	in: o número de bytes a escrever
ulMode	in: definir como CVC_WRITE_MODE_PAD para preencher o ficheiro com zeros se (ulFileOffset + inlen) for menor que o comprimento do ficheiro

```
PTEIDLIB_API long
PTEID_CVC_WriteSOD
                        ( unsigned long            ulFileOffset,
                          const unsigned char *    in,
                          unsigned long            inlen,
                          unsigned long            ulMode
                        )
```

Esta função chama PTEID_CVC_WriteFile() com o ficheiro SOD file como caminho.

Parâmetros:

ulFileOffset	in: escolher qual o offset no ficheiro por onde iniciar a escrita
in	in: os conteúdos do ficheiro

inlen	in: o número de bytes a escrever
ulMode	in: definir como CVC_WRITE_MODE_PAD para preencher o ficheiro com zeros se (ulFileOffset + inlen) for menor que o comprimento do ficheiro

PTEIDLIB_API long PTEID_Exit(unsigned long *ulMode*)

Limpa todos os dados usados pelo toolkit. Esta função tem que ser chamada no final do programa.

Parâmetros:

ulMode in: modo de saída, PTEID_EXIT_LEAVE_CARD ou PTEID_EXIT_UNPOWER

PTEIDLIB_API long PTEID_GetAddr(PTEID_ADDR * *AddrData* **)**

Ler os dados de Morada:

Parâmetros:

AddrData out: o endereço do registo PTEID_ADDR

PTEIDLIB_API long PTEID_GetCertificates (PTEID_Certifs * *Certifs* **)**

Ler todos os certificados pessoais e de CA

Parâmetros:

Certifs out: o endereço do registo PTEID_Certifs

PTEIDLIB_API long PTEID_GetCVCRoot (PTEID_RSAPublicKey * *pCVCRootKey* **)**

Obter a chave pública CVC CA que este cartão utilize para verificar a chave CVC; permitir que a aplicação seleccione o certificado CVC correcto para este cartão.

Não será alocada nenhuma memória para o registo PTEID_RSAPublicKey struct por isso os campos "modulus" e "exponent" deverão ter memória suficiente alocada para guardar os respectivos valores; e a quantidade de memória deve ser dada nos campos "Lenght".

Por exemplo:

```
unsigned char modulus[128];
unsigned char exponent[3];
PTEID_RSAPublicKey CVCRootKey = {modulus, sizeof(modulus), exponent,
```

sizeof(exponent)};

Após retorno bem sucedido, os campos modulusLength e exponentLenght irão conter os correctos modulus length resp. exponent length.

Parâmetros:

pCVCRootKey in: o endereço do registo PTEID_RSAPublicKey

PTEIDLIB_API long PTEID_GetID(PTEID_ID * ***IDData*** **)**

Ler os dados de Identificação

Parâmetros

IDData out: o endereço do registo PTEID_ID

PTEIDLIB_API long PTEID_GetPic(PTEID_PIC * ***PicData*** **)**

Ler a fotografia.

Parâmetros:

PicData out: o endereço do registo PTEID_PIC

PTEIDLIB_API long PTEID_GetPINs(PTEID_Pins * ***Pins*** **)**

Devolver os PINs (listados nos ficheiros PKCS15).

Parâmetros:

Pins out: o endereço do registo PTEID_Pins

PTEIDLIB_API long PTEID_GetTokenInfo(PTEID_TokenInfo * ***tokenData*** **)**

Devolver os conteúdos de PKCS15 TokenInfo.

Parâmetros:

tokenData out: o endereço do registo PTEID_TokenInfo

PTEIDLIB_API long PTEID_Init(char * ***ReaderName*** **)**

Inicializa o toolkit.

Esta função deve ser chamada antes de qualquer outra; tenta efectuar uma ligação ao cartão e caso não esteja inserido nenhum cartão é devolvido um erro. Quando o

cartão é removido do leitor, esta função tem que ser chamada novamente.

Parâmetros:

ReaderName in: o nome do leitor PCSC (tal como devolvido por SCardListReaders()), especifique NULL se quiser seleccionar o primeiro leitor

PTEIDLIB_API long PTEID_IsActivated(unsigned long * *pulStatus*)

Obter o estado de activação do cartão.

Parâmetros:

pulStatus out: o estado de activação: 0 se não estiver activo, 1 se activado

```
PTEIDLIB_API long
PTEID_ReadFile
                    ( unsigned char *       file,
                      int                   filelen,
                      unsigned char *       out,
                      unsigned long *       outlen,
                      unsigned char         PinId
                    )
```

Ler um ficheiro no cartão.

Se uma referência PIN é fornecida e necessária para ler o ficheiro, o PIN será solicitado e verificado se necessário. Se *outlen for menor que os conteúdos do ficheiro, apenas os bytes *outlen serão lidos. Se *outlen for maior os conteúdos do ficheiro são devolvidos sem erro.

Parâmetros:

file in: um byte array contendo o caminho do ficheiro, como por exemplo {0x3F, 0x00, 0x5F, 0x00, 0xEF, 0x02}, para o ficheiro ID

filelen in: comprimento do ficheiro

out out: o buffer que guarda os conteúdos do ficheiro

outlen in/out: número de bytes alocados / número de bytes lidos

PinId in: o ID do PIN de Morada (apenas necessário aquando da leitura do ficheiro de Morada)

```
PTEIDLIB_API long
PTEID_ReadSOD
                    ( unsigned char *       out,
                      unsigned long *       outlen
                    )
```

Ler os conteúdos do ficheiro SOD a partir do cartão.

Esta função chama PTEID_ReadFile() com o ficheiro SOD como caminho. Se *outlen for menor que os conteúdos do ficheiro, apenas os bytes *outlen serão lidos. Se *outlen for maior os conteúdos do ficheiro são devolvidos sem erro.

Parâmetros:

out out: o buffer para guardar os conteúdos do ficheiro
outlen in/out: número de bytes alocados / número de bytes lidos

```
PTEIDLIB_API long PTEID_SelectADF( unsigned char * adf,
                                   long adflen )
```

Seleccionar um Application Directory File (ADF) através da AID (Application ID).

Parâmetros

adf in: a AID da ADF
adflen in: o comprimento

```
PTEIDLIB_API long PTEID_SetSODCAs( PTEID_Certifs * Certifs )
```

Especificar os certificados (raiz) que são usados para assinar os certificados DocumentSigner no ficheiro SOD.

(O ficheiro SOD no cartão está assinado por um certificado Document Signer, e este certificado está também no SOD)

Por omissão, esta biblioteca lê os certificados que estão presentes no directório *%appdir%*/eidstore/certs (%appdir% corresponde ao directório onde a aplicação reside). Se este directório não existir (ou não contiver os certificados correctos para o cartão), deverá chamar esta função para o especificar; ou desactivar a verificação SOD com a função PTEID_SetSODChecking(). Se chamar esta função novamente com o parâmetro NULL, os certificados *default* serão novamente usados.

Parâmetros:

Certifs in: o endereço de PTEID_Certifs, ou NULL

```
PTEIDLIB_API long PTEID_SetSODChecking( int bDoCheck )
```

Activar / Desactivar a verificação SOD.

Verificação "SOD" significa que a validade dos dados de identificação, morada, fotografia e chave pública de autenticação do cartão, é verificada de modo a assegurar que não é falsificada. Este processo é efectuado através da leitura do ficheiro SOD que contém hashes sobre os dados mencionados acima e está assinada por um certificado DocumentSigner.

Parâmetros:

bDoCheck in: *true* para activar verificação SOD checking, *false* para desactivar

```
PTEIDLIB_API long
PTEID_UnblockPIN
                        (unsigned char  PinId,
                         char *         pszPuk,
                         char *         pszNewPin,
                         long *         triesLeft
                        )
```

Desbloquear PIN com alteração de PIN.

Se pszPuk == NULL ou pszNewPin == NULL, uma caixa de diálogo é mostrada solicitando o PUK e o novo PIN

Parâmetros:

PinId	in: o PIN ID, ver o registo PTEID_Pins
pszPuk	in: o valor PUK, se NULL será solicitado o PUK ao utilizador.
pszNewPin	in: o novo PIN, se NULL será solicitado o PIN ao utilizador.
triesLeft	out: o número restante de tentativas PUK

```
PTEIDLIB_API long
PTEID_UnblockPIN_Ext
                        (unsigned char  PinId,
                         char *         pszPuk,
                         char *         pszNewPin,
                         long *         triesLeft,
                         unsigned long  ulFlags
                        )
```

Funcionalidade estendida de desbloqueio de PIN

Ex: chamar PTEID_UnblockPIN_Ext() com ulFlags = UNBLOCK_FLAG_NEW_PIN é o mesmo que chamar PTEID_UnblockPIN(...)

Parâmetros:

PinId	in: o PIN ID, ver o registo PTEID_Pins
pszPuk	in: o valor PUK, se NULL será solicitado o PUK ao utilizador.

pszNewPin in: o novo PIN, se NULL será solicitado o PIN ao utilizador.
triesLeft out: o número restante de tentativas PUK
ulFlags in: flags: 0, UNBLOCK_FLAG_NEW_PIN, UNBLOCK_FLAG_PUK_MERGE ou UNBLOCK_FLAG_NEW_PIN | UNBLOCK_FLAG_PUK_MERGE

```
PTEIDLIB_API long
PTEID_VerifyPIN          (unsigned char   PinId,
                          char *          Pin,
                          long *          triesLeft
                         )
```

Verificar um PIN.

Parâmetros:

PinId in: o PIN ID, ver o registo PTEID_Pins
Pin in: o valor PIN, se NULL será solicitado o PIN ao utilizador.
triesLeft out: o número restante de tentativas PIN

```
PTEIDLIB_API long
PTEID_WriteFile          (unsigned char *  file,
                          int              filelen,
                          unsigned char *  in,
                          unsigned long    inlen,
                          unsigned char    PinId
                         )
```

Escrever dados para um ficheiro no cartão.

Se for fornecida uma referência PIN, este será solicitado e verificado se necessário (verificação "just-in-time"). Esta função aplica-se apenas a escrita no ficheiro Personal Data.

Parâmetros:

file in: um array de bytes, contendo o caminho para o ficheiro. Ex: {0x3F, 0x00, 0x5F, 0x00, 0xEF, 0x02} para o ficheiro ID
filelen in: comprimento do ficheiro
in in: os dados a ser escritos no ficheiro
inlen in: o comprimento dos dados a escrever
PinId in: o ID do PIN de Autenticação, ver o registo PTEID_Pins

4.10. Caching de ficheiros

Devido ao facto da leitura de ficheiros do cartão ser um processo demorado, especialmente em leitores mais lentos, certos ficheiros são guardados no disco rígido quando são lidos pela primeira vez:

- certos ficheiros pkcs15
- o ficheiro ID
- o ficheiro SOD

O ficheiro de morada não é cached pois pode vir a mudar e porque contém dados protegidos por PIN.

O ficheiro SOD pode também vir a mudar por isso uma pequena parte do SOD é lido do cartão para verificar se a SOD que foi cached ainda está actualizada. Caso não esteja, o ficheiro completo é lido do cartão e copiado novamente para o disco rígido.

Nota para CVC: existe uma flag no middleware que regista quando o SOD foi lido e, por exemplo, quando efectua um GetID() e depois GetAddress(), o SOD não é lido novamente para verificar a validade dos dados de Morada. Como excepção, quando CVC_Write_XXX() é chamado, é feito um reset à flag.

4.11. Códigos de Erro

Existem 2 tipos de return codes: os do próprio pteidlib, e os da biblioteca open-source subjacente pteidlibopensc.

Return codes da biblioteca pteidlib:

#define PTEID_OK	0 /* Function succeeded */
#define PTEID_E_BAD_PARAM	1 /* Invalid parameter (NULL pointer, out of bound, etc.) */
#define PTEID_E_INTERNAL	2 /* An internal consistency check failed */
#define PTEID_E_INSUFFICIENT_BUFFER	3 /* The data buffer to receive returned data is too small */
#define PTEID_E_KEYPAD_CANCELLED	4 /* Input on pinpad cancelled */
#define PTEID_E_KEYPAD_TIMEOUT	5 /* Timout returned from pinpad */
#define PTEID_E_KEYPAD_PIN_MISMATCH	6 /* The two PINs did not match */
#define PTEID_E_KEYPAD_MSG_TOO_LONG	7 /* Message too long on pinpad */

#define PTEID_E_INVALID_PIN_LENGTH 8 /* Invalid PIN length */

#define PTEID_E_NOT_INITIALIZED 9 /* Library not initialized */

#define PTEID_E_UNKNOWN 10 /* An internal error has been detected, but the source is unknown */

Return codes da biblioteca open-source subjacente pteidlibopensc - Licença LGPL:

/* Erros relativos a operações do leitor */

#define SC_ERROR_READER -1100

#define SC_ERROR_NO_READERS_FOUND -1101

#define SC_ERROR_SLOT_NOT_FOUND -1102

#define SC_ERROR_SLOT_ALREADY_CONNECTED -1103

#define SC_ERROR_CARD_NOT_PRESENT -1104

#define SC_ERROR_CARD_REMOVED -1105

#define SC_ERROR_CARD_RESET -1106

#define SC_ERROR_TRANSMIT_FAILED -1107

#define SC_ERROR_KEYPAD_TIMEOUT -1108

#define SC_ERROR_KEYPAD_CANCELLED -1109

#define SC_ERROR_KEYPAD_PIN_MISMATCH -1110

#define SC_ERROR_KEYPAD_MSG_TOO_LONG -1111

#define SC_ERROR_EVENT_TIMEOUT -1112

#define SC_ERROR_CARD_UNRESPONSIVE -1113

#define SC_ERROR_READER_DETACHED -1114

#define SC_ERROR_READER_REATTACHED -1115

/* Resultantes de um comando de cartão ou relacionado com o cartão */

#define SC_ERROR_CARD_CMD_FAILED -1200

#define SC_ERROR_FILE_NOT_FOUND -1201

#define SC_ERROR_RECORD_NOT_FOUND -1202

#define SC_ERROR_CLASS_NOT_SUPPORTED -1203

#define SC_ERROR_INS_NOT_SUPPORTED -1204

#define SC_ERROR_INCORRECT_PARAMETERS -1205

#define SC_ERROR_WRONG_LENGTH -1206

#define SC_ERROR_MEMORY_FAILURE -1207

#define SC_ERROR_NO_CARD_SUPPORT -1208

#define SC_ERROR_NOT_ALLOWED -1209

#define SC_ERROR_INVALID_CARD -1210

#define SC_ERROR_SECURITY_STATUS_NOT_SATISFIED -1211

#define SC_ERROR_AUTH_METHOD_BLOCKED -1212

```
#define SC_ERROR_UNKNOWN_DATA_RECEIVED      -1213
#define SC_ERROR_PIN_CODE_INCORRECT         -1214
#define SC_ERROR_FILE_ALREADY_EXISTS        -1215

/* Devolvidos pela biblioteca OpenSC quando invocada com argumentos inválidos */
#define SC_ERROR_INVALID_ARGUMENTS          -1300
#define SC_ERROR_CMD_TOO_SHORT              -1301
#define SC_ERROR_CMD_TOO_LONG               -1302
#define SC_ERROR_BUFFER_TOO_SMALL           -1303
#define SC_ERROR_INVALID_PIN_LENGTH         -1304

/* Resultantes de operações internas da biblioteca OpenSC */
#define SC_ERROR_INTERNAL                   -1400
#define SC_ERROR_INVALID_ASN1_OBJECT        -1401
#define SC_ERROR_ASN1_OBJECT_NOT_FOUND      -1402
#define SC_ERROR_ASN1_END_OF_CONTENTS       -1403
#define SC_ERROR_OUT_OF_MEMORY              -1404
#define SC_ERROR_TOO_MANY_OBJECTS           -1405
#define SC_ERROR_OBJECT_NOT_VALID           -1406
#define SC_ERROR_OBJECT_NOT_FOUND           -1407
#define SC_ERROR_NOT_SUPPORTED              -1408
#define SC_ERROR_PASSPHRASE_REQUIRED        -1409
#define SC_ERROR_EXTRACTABLE_KEY            -1410
#define SC_ERROR_DECRYPT_FAILED             -1411
#define SC_ERROR_WRONG_PADDING              -1412
#define SC_ERROR_WRONG_CARD                 -1413

/* Relacionados com o PKCS #15 init */
#define SC_ERROR_PKCS15INIT                 -1500
#define SC_ERROR_SYNTAX_ERROR               -1501
#define SC_ERROR_INCONSISTENT_PROFILE       -1502
#define SC_ERROR_INCOMPATIBLE_KEY           -1503
#define SC_ERROR_NO_DEFAULT_KEY             -1504
#define SC_ERROR_ID_NOT_UNIQUE              -1505
#define SC_ERROR_CANNOT_LOAD_KEY            -1006

/* Erros que não se enquadram nas categorias acima */
#define SC_ERROR_UNKNOWN                    -1900
#define SC_ERROR_PKCS15_APP_NOT_FOUND       -1901
```