Projeto Cartão de Cidadão

Manual técnico do *Middleware* Cartão de Cidadão

Outubro 2015

Versão 1.60.0

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 direcionada à funcionalidade não-criptográfica do Cartão de Cidadão.

No Windows, estas bibliotecas podem ser encontradas na pasta *System32*; no Mac OS X e em Linux na diretoria */usr/local/lib*.

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 respetivos.

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
- Verificar em-linha a validade dos certificados
- Registar os certificados do governo e do cidadão (apenas Windows XP)
- Gestão de códigos PIN (Testar o PIN, Alterar o PIN). Precisa de ligação à Internet para alterar o PIN de autenticação.
- Gerir a Pasta Pessoal
- Ver informação específica de ministérios
- Imprimir a informação do cartão para PDF ou impressora
- Alterar a morada no cartão. Precisa de ligação à Internet.

Tray applet

Esta aplicação é instalada como um ícone da área de notificação. No Windows, aparece normalmente no canto inferior direito do ecrã, no Mac no canto superior direito do ecrã. Quando ativada (o utilizador pode desativá-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 ativada) 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 ativada). Este modo é também conhecido como modo "Kiosk". Esta funcionalidade a nível de certificados é apenas implementada na plataforma Windows.

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

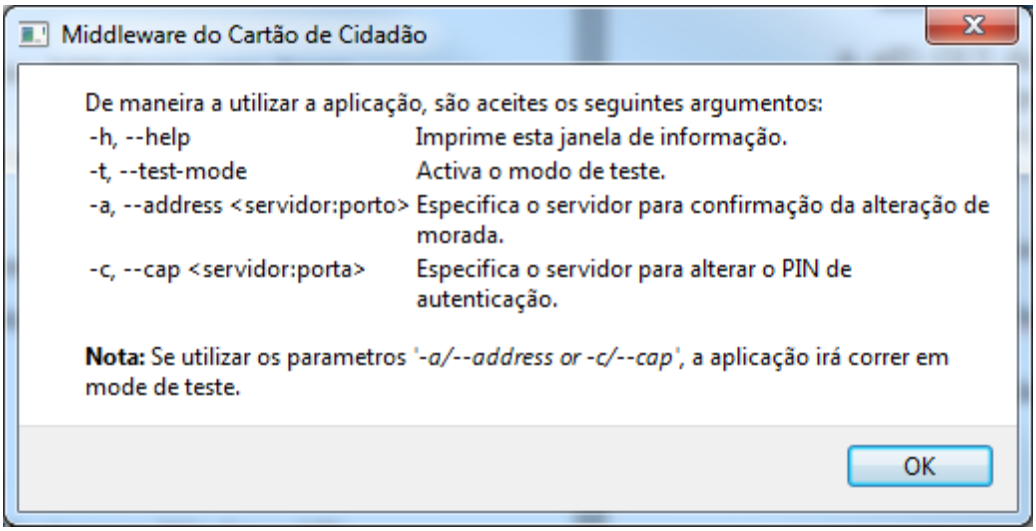
1.1. Modo de teste

A eID GUI dispõe de um modo de funcionamento designado *modo de teste* que possui algumas facilidades para programadores e técnicos. Para iniciar em modo de teste, a aplicação *pteidgui.exe* (Windows) ou *pteidgui* (Linux/Mac OS X) deve ser invocada a partir da linha de comando.

Para obter uma lista dos argumentos, utilizar o seguinte comando:

pteidgui –help

A invocação deste comando faz aparecer a seguinte janela de ajuda:



Através dos argumentos é possível alterar os endereços dos servidores de Alteração de Morada ou Alteração de CAP PIN.

Caso não seja especificado um ou ambos os endereços, serão utilizados os endereços de teste por omissão:

Servidor de alteração de morada:
pki.teste.cartaodecidadao.pt:443

Servidor de alteração de CAP PIN:
paf.teste.cartaodecidadao.pt:443

NOTA: Tal como no modo normal, para poder ler um cartão de testes, os certificados correspondentes deverão ser copiados para a diretoria *%APPDIR%/eidstore/certs* (*%APPDIR %* corresponde à diretoria onde a aplicação reside) ao cartão.

Quando invocada em modo de teste, a aplicação apresenta em fundo uma marca de água com a palavra "Teste", para indicar ao utilizador que está em modo de teste e não em utilização normal.

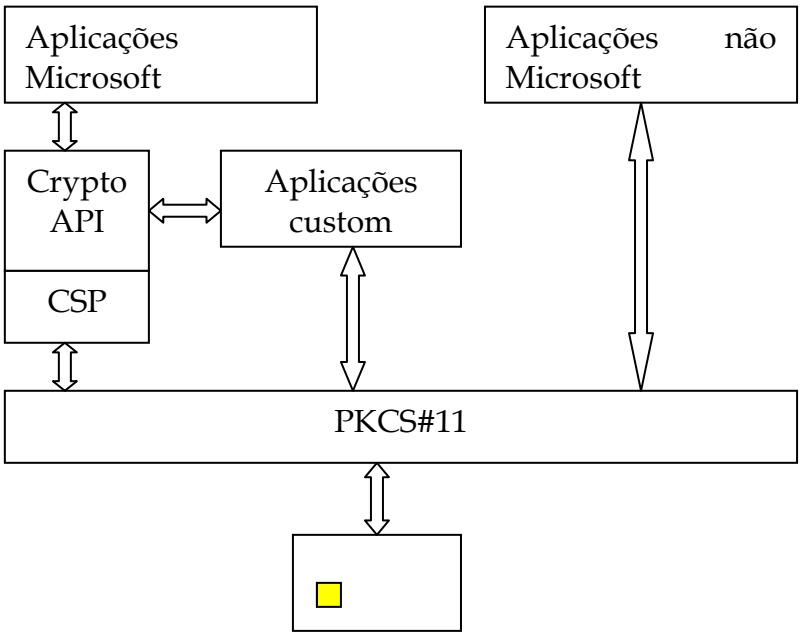
2. Documentação CSP

2.1. Introdução

Para as aplicações Microsoft® (Office, Outlook...) é criado um *Cryptographic Service Provider* (CSP) que implementa as operações criptográficas do *smartcard*. Uma aplicação nunca chamará esta implementação diretamente mas sim através de uma interface Microsoft® chamada Crypto API.

A segunda interface implementada, PKCS#11, 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, assinatura e cifra 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 CryptoAPI e a interface PKCS#11 subjacente. O programador nunca irá chamar nenhuma das funções do CSP diretamente 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-repúdio (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 correto 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 Microsoft®. Chamar funções da CryptoAPI 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 cifra 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*);

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 ()**.

2.2.8. 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 exceção 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 arriscada.

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 correto 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 as 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 cifra. 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 cifra 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 cifra. 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 cifra 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 (exceto 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 efetuada 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 retro compatibilidade, 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 Firefox. Aplicações desenvolvidas podem recorrer a este interface em vez do interface CryptoAPI. A interface PKCS#11 é por vezes chamada Cryptoki.

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

3.1.1. Diferenças na autenticação

Ao contrário da filosofia original do PKCS#11, não existe no Cartão de Cidadão o conceito de "login". Existem dois PINs que deverão ser verificados antes da utilização das respetivas operações criptográficas: O PIN de autenticação e o PIN de assinatura.

Os programadores que desejem tirar partido da implementação do PKCS#11 para o Cartão de Cidadão deverão ter em conta uma particularidade: O Método C_Login não deve ser usado para verificação de PINs. Em lugar disso, a utilização das operações de assinatura desencadeia automaticamente o pedido de PIN ao utilizador através de caixas de diálogo ou *pinpad*.

3.1.2. Chamadas à API implementadas

3.1.2.1. Funções Gerais

C_Initialize,

C_Finalize

C_GetInfo

C_GetFunctionList

3.1.2.2. Funções de gestão de *slot* e *token*

C_GetSlotList

C_GetSlotInfo

C_GetTokenInfo

C_GetMechanismList

C_GetMechanismInfo

C_WaitForSlotEvent (*only non-blocking*)

C_SetPin

3.1.2.3. Funções de gestão de sessão

C_OpenSession

C_CloseSession

C_CloseAllSessions

C_GetSessionInfo

C_Login

C_Logout

3.1.2.4. Funções de gestão de objetos

C_FindObjectsInit

C_FindObjects

C_FindObjectsFinal

C_GetAttributeValue

3.1.2.5. Funções de assinatura

C_SignInit

C_Sign

C_SignUpdate

C_SignFinal

3.1.2.6. Funções de *digest*

C_DigestInit

C_Digest

C_DigestUpdate

C_DigestFinal

3.1.2.7. Funções de geração aleatória (a aguardar confirmação)

C_SeedRandom

C_GenerateRandom

3.1.3. Mecanismos de assinatura suportados

Para assinaturas:

- CKM_RSA_PKCS: both ASN.1-wrapped e *hashes* puros (MD5, SHA1, SHA1+MD5, SHA256, RIPEMD160)
- CKM_RIPEMD160_RSA_PKCS, CKM_SHA1_RSA_PKCS, CKM_SHA256_RSA_PKCS, CKM_MD5_RSA_PKCS

Para *digests*:

- CKM_SHA_1, CKM_RIPEMD160, CKM_MD5

3.1.4. 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 objeto CKA_ID.

3.1.5. 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

O SDK está destinado a organizações cujo objetivo 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 direcionado 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 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 5 categorias:

- *Funções de **inicialização** e **término***, obrigatórias para iniciar e terminar a utilização do *toolkit*
- *Funções de **identity***, usadas para obter dados identificativos (nome, morada, etc.) do cartão
- *Funções de **uso geral 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.
- *Funções de **segurança***, Existem também funções para garantir a integridade dos dados do cartão. Existem também funções para estabelecer sessões seguras sessões com garantia de autenticação e confidencialidade.
- *Funções de **alteração do cartão***, são funções que permitem alterar a morada escrita no cartão e o PIN da *applet* EMV-CAP.

4.4.1. Funções de inicialização e término

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

Funções:

- PTEID_Init()
- PTEID_Exit()

4.4.2. Funções Identidade

Todas as funções de Identidade são autossuficientes. Isto é, não é necessário chamar nenhuma outra função em conjunto com uma função identidade (excerto as de inicialização e término). Não é necessário inserir um PIN para ler ficheiros de identidade. Todas estas funções podem ser chamadas independentemente do estado presente do cartão – caso outra DF (Data File) que não a de identidade estiver selecionada, 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()
- PTEID_GetCVCRoot()

4.4.3. Funções genéricas de alto nível

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

Funções

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

4.4.4. Funções de segurança

4.4.4.1. Autenticidade dos dados do cartão

Para assegurar que os dados no cartão são genuínos, um ficheiro SOD (*Security Object Data*) é colocado no cartão contendo assinaturas eletrónicas 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_ReadSOD()
- PTEID_SetSODChecking()
- PTEID_SetSODCAs()

4.4.4.2. Sessões seguras

O cartão permite o estabelecimento de sessões seguras. É efetuada a autenticação de uma ou ambas as partes (a aplicação e o cartão). Após este processo as operações seguintes são efetuadas sobre comunicação cifrada e autenticada.

A autenticação da aplicação é efetuada através de CVCs (*Card Verifiable Certificates*). Estes certificados são emitidos para entidades que estejam autorizadas a efetuar operações privilegiadas no cartão. Existem duas operações privilegiadas que obrigam ao estabelecimento prévio de uma sessão segura:

- Leitura da morada sem introdução de PIN.
- Alteração da morada.

Existem duas versões do cartão atualmente em utilização, 0.7 e 1.0.1. O estabelecimento de sessões seguras é bastante diferente de uma versão para outra.

No caso de um cartão 0.7. as funções são as seguintes:

- PTEID_CVC_R_Init()
- PTEID_CVC_R_DH_Auth()
- PTEID_CVC_R_ValidateSignature()
- PTEID_CVC_Authenticate()

No caso de um cartão 1.0.1 as funções são as seguintes:

- PTEID_CVC_Init()
- PTEID_CVC_R_ValidateSignature()
- PTEID_CVC_Init_SM101()
- PTEID_CVC_Authenticate_SM101()

4.4.4.3. Todas as funções de segurança

- PTEID_GetCardAuthenticationKey()
- PTEID_CVC_Init()
- PTEID_CVC_Authenticate()
- PTEID_CVC_Init_SM101()
- PTEID_CVC_Authenticate_SM101()
- PTEID_CVC_ReadFile()
- PTEID_CVC_WriteFile()
- PTEID_CVC_GetAddr()
- PTEID_CVC_WriteAddr()
- PTEID_CVC_WriteSOD()
- PTEID_CVC_R_Init()
- PTEID_CVC_R_DH_Auth()
- PTEID_CVC_R_ValidateSignature()

4.4.5. Funções de alteração do cartão

O utilizador tem acesso a duas operações de alteração do cartão, são estas a alteração da morada e do PIN da *applet* EMV-CAP. Ambas as operações precisam de uma ligação à Internet para estabelecer uma ligação segura entre o servidor e o cartão.

As funções que permitem a alteração de dados do cartão são:

- PTEID_ChangeAddress()
- PTEID_GetChangeAddressProgress()
- PTEID_SetChangeAddressCallback()
- PTEID_CancelChangeAddress();
- PTEID_CAP_ChangeCapPin()
- PTEID_CAP_GetCapPinChangeProgress()
- PTEID_CAP_SetCapPinChangeCallback()
- PTEID_CAP_CancelCapPinChange()
- PTEID_GetLastWebErrorCode()
- PTEID_GetLastWebErrorMessage()

4.5. Detalhes da API C/C++

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

Para a API *managed* C++, por favor consulte os ficheiros *header* (*.h) na diretoria "*dotnet*" do SDK.

A API .NET está implementada na biblioteca *pteidlib_dotnet.dll* na diretoria **System32** do Windows.

Para a API Java, por favor consulte os Javadocs na diretoria "*java*" do SDK.

4.5.1. PTEID_Activate

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

Ativar o cartão (= atualizar um ficheiro específico do cartão).

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

Parâmetros:

pszPin

in: valor do PIN de Ativação

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 Ativação, ou 0 para o inverso (deve apenas ser usado para testes)

4.5.2. PTEID_CAP_CancelCapPinChange()

void PTEID_CAP_CancelCapPinChange()

Cancelar uma operação de alteração do PIN CAP que esteja em curso.

4.5.3. PTEID_CAP_ChangeCapPin

long PTEID_CAP_ChangeCapPin(
const char *csServer,
const unsigned char *ucServerCaCert,
unsigned long ulServerCaCertLen,
tProxyInfo *proxyInfo,
const char *pszOldPin,
const char *pszNewPin,
long *triesLeft)

Iniciar uma alteração do PIN CAP. A função devolve códigos de erro genéricos. Para saber mais detalhes sobre um erro podem ser utilizadas as funções PTEID_CAP_GetCapPinChangeProgress, PTEID_GetLastWebErrorCode e PTEID_GetLasWebErrorMessage.

Parâmetros:

csServer
in: Endereço do servidor, no formato: <nome>:<porto>.
ucServerCaCert
in: Certificado da AC do servidor, codificado em DER.
ulServerCaCertLen
in: Comprimento do certificado da AC do servidor.
proxyInfo
in: Dados do servidor proxy, ou NULL se não for necessário.
pszOldPin
in: O PIN CAP actual.
pszNewPin
in: O novo PIN CAP que se pretende.
triesLeft
out: O número de tentativas restantes para alterar o PIN CAP.

4.5.4. PTEID_CAP_GetCapPinChangeProgress

tCapPinChangeState PTEID_CAP_GetCapPinChangeProgress()

Permite saber o estado de uma operação de alteração do PIN CAP enquanto esta decorre. Os valores possíveis de retorno são:

CAP_INITIALISING
A inicializar a alteração de PIN.

CAP_CONNECTING
A ligar ao servidor.
CAP_READING_INFO
A ler informação do cartão.
CAP_SENDING_INFO
A enviar os dados para o servidor.
CAP_WRITE
A escrever os dados no cartão
CAP_FINISH
A terminar a transação (enviar resultado para o servidor).
CAP_FINISHED
A transação está completa.
CAP_CANCELLED
A transação foi cancelada.

4.5.5. PTEID_CAP_SetCapPinChangeCallback

void PTEID_CAP_SetCapPinChangeCallback(
void(_USERENTRY * callback)(tCapPinChangeState state))

Atribui ao middleware uma função de *callback*, definida pelo utilizador, para ser chamada durante a alteração do PIN CAP. Esta função é invocada pelo middleware de cada vez que o estado da operação se altera.

4.5.6. PTEID_CVC_Authenticate

long PTEID_CVC_Authenticate(
unsigned char *pucSignedChallenge,
int iSignedChallengeLen)

Passo final do estabelecimento de uma sessão segura num cartão de versão 0.7. Envia ao cartão o *challenge* assinado pela aplicação para autenticação desta.

Parâmetros:
pucSignedChallenge
in: O *challenge* assinado pela chave privada correspondente ao CVC da aplicação.
iSignedChallengeLen
in: O comprimento da assinatura, deve ser 128.

4.5.7. PTEID_CVC_Authenticate_SM101

long PTEID_CVC_Authenticate_SM101(
const unsigned char *ifdChallenge,
int ifdChallengeLen,
const char *ifdSerialNr,
int ifdSerialNrLen,
const char *iccSerialNr,
int iccSerialNrLen,
const unsigned char *keyIfd,
int keyIfdLen,
const unsigned char * encKey,
unsigned int encKeyLen,
const unsigned char * macKey,
unsigned int macKeyLen,
unsigned char *ifdChallengeResp,
int * ifdChallengeRespLen)

Passo final do estabelecimento de uma sessão segura num cartão de versão 1.0.1. Neste passo o cartão e a aplicação autenticam-se mutuamente utilizando um sistema de chave simétrica.

Parâmetros:

ifdChallenge
in: O *challenge* que foi assinado pela chave derivada da aplicação.

ifdChallengeLen
in: O comprimento do *challenge* assinado, deve ser 48.

ifdSerialNr
in: O número de série da aplicação.

ifdSerialNrLen
in: O comprimento do número de série da aplicação.

iccSerialNr
in: O número de série do cartão.

iccSerialNrLen
in: O comprimento do número de série do cartão.

keyIfd
in: A chave secreta kIFD gerada pela aplicação

keyIfdLen
in: O comprimento da chave secreta kIFD.

encKey
in: A chave derivada a ser usada para cifra.

encKeyLen
in: O comprimento da chave derivada de cifra.

macKey
in: A chave derivada a usar para efetuar MAC ou seja, autenticação de mensagens.
macKeyLen
in: O comprimento da chave derivada de MAC.
ifdChallengeResp
out: A resposta do cartão ao pedido de autenticação.
ifdChallengeRespLen
out: O comprimento da resposta do cartão, deve ser pelo menos 48.

4.5.8. PTEID_CVC_GetAddr

long PTEID_CVC_GetAddr(PTEID_ADDR *AddrData)

Ler o ficheiro de morada sobre uma sessão segura e colocar os conteúdos num registo PTEID_ADDR. Dispensa a introdução do PIN de morada.
Uma sessão segura terá que ser estabelecida previamente. Isto depende da versão do cartão. No caso 0.7 uma autenticação CVC é suficiente. No caso 1.0.1 é necessário estabelecer uma autenticação CVC e uma autenticação mútua com chave simétrica para estabelecer um canal seguro (*Secure Messaging*).
Parâmetros:
AddrData
out: o endereço de um registo PTEID_ADDR

4.5.9. PTEID_CVC_Init

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

Iniciar o estabelecimento de uma sessão segura com um cartão de versão 1.0.1. Este é o primeiro passo, inicia a autenticação CVC.

Parâmetros:
pucCert
in: O CVC da aplicação.
iCertLen
in: O comprimento do CVC.
pucChallenge
out: O *challenge* que o cartão envia para a aplicação assinar.

iChallengeLen

in: O tamanho reservado para o *challenge*. Deve ser pelo menos 128.

4.5.10. PTEID_CVC_Init_SM101

long PTEID_CVC_Init_SM101(
unsigned char *pucChallenge,
int iChallengeLen)

Inicia o estabelecimento de uma sessão cifrada com um cartão de versão 1.0.1. Esta função deve ser usada para iniciar uma autenticação mútua depois de uma autenticação baseada em CVC ter sido concluída.

Parâmetros:

pucChallenge

out: O *challenge* gerado pelo cartão para ser assinado pela chave simétrica.

iChallengeLen

in: O tamanho do espaço reservado para o *challenge*, deve ser pelo menos 48.

4.5.11. PTEID_CVC_R_DH_Auth

long PTEID_CVC_R_DH_Auth(
const unsigned char *ucKifd,
unsigned long ulKifdLen,
const unsigned char *pucCert
unsigned long ulCertLen
unsigned char *ucKicc
unsigned long *ulKiccLen
unsigned char *ucChallenge
unsigned long *ulChallengeLen)

Segundo passo no estabelecimento de uma sessão segura com um cartão de versão 0.7. A aplicação autentica-se perante o cartão neste passo.

Parâmetros:

ucKifd

in: Chave Kifd, gerada pela aplicação.

ulKifdLen

in: Comprimento da chave Kifd, deve ser 128.

pucCert

in: CVC correspondente à aplicação.

ulCertLen
in: Comprimento do CVC.
ucKicc
out: Chave Kicc, gerada pelo cartão.
ulKiccLen
out: Comprimento da chave Kicc, deve ser 128.
ucChallenge
out: O *challenge* para a aplicação assinar e autenticar-se.
ulChallengeLen
out: Comprimento do *challenge*, deve ser 128.

4.5.12. PTEID_CVC_R_Init

long PTEID_CVC_R_Init(
unsigned char *ucG,
unsigned long *outlenG,
unsigned char *ucP,
unsigned long *outlenP,
unsigned char *ucQ,
unsigned long *outlenQ)

Inicia o estabelecimento de uma sessão segura num cartão de versão 0.7. Faz a leitura dos parâmetros para o algoritmo Diffie-Hellman de troca de chaves.

Parâmetros:
ucG
out: O parâmetro G do algoritmo Diffie-Hellman.
outlenG
in/out: Comprimento do parâmetro G, deve ser 128.
ucP
out: O parâmetro P do algoritmo Diffie-Hellman.
outlenP
in/out: Comprimento do parâmetro P, deve ser 128.
ucQ
out: O parâmetro Q do algoritmo Diffie-Hellman.
outlenQ
in/out: Comprimento do parâmetro Q, deve ser 20.

4.5.13. PTEID_CVC_R_ValidateSignature

long PTEID_CVC_R_ValidateSignature(
const unsigned char *pucSignedChallenge,
unsigned long ulSignedChallengeLen)

Segundo passo no estabelecimento de uma sessão segura com um cartão de versão 1.0.1. A aplicação envia para o cartão o *challenge* assinado para concluir o passo de autenticação com CVC.

Parâmetros:

pucSignedChallenge
in: O challenge assinado pela aplicação.
ulSignedChallengeLen
in: O comprimento do challenge assinado, deve ser 128.

4.5.14. PTEID_CVC_ReadFile

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

Ler os conteúdos de um ficheiro sobre uma sessão segura.

O estabelecimento de uma sessão segura terá de ser efetuado 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 comprimento 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.

4.5.15. PTEID_CVC_WriteAddr

long PTEID_CVC_WriteAddr(const PTEID_ADDR *AddrData)

Escrever o ficheiro de morada sobre uma sessão segura. O estabelecimento de uma sessão segura terá de ser efetuado anteriormente.

Parâmetros:

AddrData

in: o endereço de um registo PTEID_ADDR

4.5.16. PTEID_CVC_WriteFile

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 uma sessão segura. O estabelecimento de uma sessão segura terá de ser efetuado 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: O conteúdo a escrever no 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.

4.5.17. PTEID_CVC_WriteSOD

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: O conteúdo a escrever no 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.

4.5.18. PTEID_CancelChangeAddress

void PTEID_CancelChangeAddress()

Cancelar uma operação de alteração de morada que esteja em curso.

4.5.19. PTEID_ChangeAddress

long PTEID_ChangeAddress(
const char *csServer,
const unsigned char *ucServerCaCert,
unsigned long ulServerCaCertLen,
tProxyInfo *proxyInfo,
const char *csSecretCode,
const char* csProcess)

Iniciar uma alteração de morada. A função devolve códigos de erro genéricos. Para saber mais detalhes sobre um erro podem ser utilizadas as funções PTEID_CAP_GetChangeAddressProgress, PTEID_GetLastWebErrorCode e PTEID_GetLasWebErrorMessage.

Parâmetros:

csServer

in: Endereço do servidor, no formato: <nome>:<porto>.

ucServerCaCert

in: Certificado da AC do servidor, codificado em DER.

ulServerCaCertLen

in: Comprimento do certificado da AC do servidor.

proxyInfo

in: Dados do servidor proxy, ou NULL se não for necessário.

csSecretCode

in: O código secreto que o cidadão recebeu por correio.

csProcess

in: O número do processo de alteração de morada, que o cidadão recebeu por correio.

4.5.20. PTEID_ChangePIN

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

Alterar um PIN.

Parâmetros:

PinId

in: O identificador do PIN, ver o registo PTEID_Pins

pszOldPin

in: O valor atual 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 tentativas restantes de validação do PIN.

4.5.21. PTEID_Exit

long PTEID_Exit(unsigned long ulMode)

Termina uma sessão e liberta o cartão.

Parâmetros:

ulMode

Modo de terminação de sessão, pode ter 2 valores:

PTEID_EXIT_LEAVE_CARD

Termina a sessão mas deixa a interface elétrica ligada.

PTEID_EXIT_UNPOWER

Termina a sessão e desliga a interface elétrica com o cartão.

4.5.22. PTEID_GetAddr

long PTEID_GetAddr(PTEID_ADDR *AddrData)

Ler os dados da Morada. Uma caixa de introdução de PIN vai ser apresentada ao utilizador.

Parâmetros:

AddrData

out: O endereço do registo PTEID_ADDR.

4.5.23. PTEID_GetCVCRoot

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 selecione o certificado CVC correto para este cartão.
Não será alocada nenhuma memória para o registo PTEID_RSAPublicKey por isso os campos "modulus" e "exponent" deverão ter memória suficiente alocada para guardar os despectivos valores; e a quantidade de memória deve ser dada nos campos "length".

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 comprimentos corretos.

Parâmetros:

pCVCRootKey

in: O endereço de um registo PTEID_RSAPublicKey.

4.5.24. PTEID_GetCardAuthenticationKey

long PTEID_GetCardAuthenticationKey(
PTEID_RSAPublicKey *pCardAuthPubKey)

Retorna a chave pública de autenticação do cartão. Esta chave serve para autenticação do cartão (mas não do cidadão) perante uma aplicação durante o estabelecimento de uma sessão segura.

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

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 comprimentos corretos.

Parâmetros:

pCardAuthPubKey

in: O endereço de um registo PTEID_RSAPublicKey.

4.5.25. PTEID_GetCardType

tCardType PTEID_GetCardType()

Devolve a versão do cartão. Os valores retornados são:

CARD_TYPE_ERR

Tipo de cartão desconhecido.

CARD_TYPE_IAS07
Cartão de versão 0.7.
CARD_TYPE_IAS101
Cartão de versão 1.0.1.

4.5.26. PTEID_GetCertificates

long PTEID_GetCertificates(PTEID_Certifs *Certifs)

Ler todos os certificados pessoais e de CA

Parâmetros:
Certifs
out: o endereço do registo PTEID_Certifs**.**

4.5.27. PTEID_GetChangeAddressProgress

tAddressChangeState PTEID_GetChangeAddressProgress()

Permite saber o estado de uma operação de alteração de morada enquanto esta decorre. Os valores possíveis de retorno são:

ADDR_INITIALISING
A inicializar a alteração de morada.
ADDR_CONNECTING
A ligar ao servidor.
ADDR_READING_INFO
A ler informação do cartão.
ADDR_SENDING_INFO
A enviar os dados para o servidor.
ADDR_INIT_SEC_CHANNEL
A inicializar o estabelecimento de canal seguro.
ADDR_SERVER_CHALL
A enviar o *challenge* para o servidor.
ADDR_SERVER_AUTH
A autenticar o servidor perante o cartão.
ADDR_CLIENT_AUTH
A autenticar o cartão perante o servidor.
ADDR_SERVER_AUTH2
A efetuar o segundo passo de autenticação.
ADDR_WRITE
A escrever a nova morada e o novo SOD no cartão.
ADDR_FINISH
A terminar a transação (enviar resultado para o servidor).

ADDR_FINISHED
A transação está completa.
ADDR_CANCELLED
A transação foi cancelada.

4.5.28. PTEID_GetID

long PTEID_GetID(PTEID_ID *IDData)

Ler os dados de Identificação do cidadão.

Parâmetros:
IDData
out: Endereço de um registo PTEID_ID.

4.5.29. PTEID_GetLastWebErrorCode

tWebErrorCode PTEID_GetLastWebErrorCode()

Permite saber o código de um erro gerado durante a alteração de PIN CAP ou morada. Esta função permite conhecer com maior detalhe um problema quando a alteração de PIN CAP ou morada falha.

Os valores possíveis de retorno são:
WEB_ERR_OK
A operação foi concluída com sucesso.
WEB_ERR_SELECT_FILE
Não foi possível encontrar um ficheiro no cartão.
WEB_ERR_READ_FILE
Ocorreu um erro a ler um ficheiro do cartão.
WEB_ERR_WRITE_FILE
Ocorreu um erro a escrever um ficheiro no cartão.
WEB_ERR_BAD_COMMAND
O cartão não compreendeu um comando ou recusa-se a executá-lo.
WEB_ERR_EMPTY_RES
A resposta do cartão tem tamanho zero.
WEB_ERR_DATA_SIZE
A resposta do cartão tem um tamanho inválido.
WEB_ERR_CARD_REMOVED
O cartão foi removido durante a operação.
WEB_ERR_CARD_COMM
O middleware não consegue comunicar com o cartão.
WEB_ERR_OUT_OF_MEM

A memória disponível para o *middleware* não é suficiente.

WEB_ERR_INTERNAL

Erro interno do *middleware*.

WEB_ERR_PARSING

Erro a interpretar uma mensagem do servidor.

WEB_ERR_MISSING_DATA

Faltam dados obrigatórios numa mensagem do servidor.

WEB_ERR_INVALID_HASH

O cartão ou o servidor enviaram um *hash* inválido.

4.5.30. PTEID_GetLastWebErrorMessage

const char * PTEID_GetLastWebErrorMessage()

Permite obter uma mensagem de erro gerado durante a alteração de PIN CAP ou morada, para mostrar ao utilizador ou guardar num ficheiro log.

4.5.31. PTEID_GetPINs

long PTEID_GetPINs(PTEID_Pins *Pins)

Devolver os PINs (listados nos ficheiros PKCS15).

Parâmetros:

Pins

out: O endereço do registo PTEID_Pins.

4.5.32. PTEID_GetPic

long PTEID_GetPic(PTEID_PIC *PicData)

Ler a fotografia.

Parâmetros:

PicData

out: O endereço do registo PTEID_PIC.

4.5.33. PTEID_GetTokenInfo

long PTEID_GetTokenInfo(PTEID_TokenInfo *tokenData)

Devolver os conteúdos de PKCS15 TokenInfo.

Parâmetros:

tokenData

out: O endereço de um registo PTEID_TokenInfo.

4.5.34. PTEID_Init

long PTEID_Init(char *ReaderName)

Inicializa o *toolkit*.

Esta função deve ser chamada antes de qualquer outra; tenta efetuar 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 selecionar o primeiro leitor disponível.

4.5.35. PTEID_IsActivated

long PTEID_IsActivated(unsigned long *pulStatus)

Obter o estado de ativação do cartão.

Parâmetros:

pulStatus

out: O estado de ativação: 0 se não estiver ativo, 1 se ativado.

4.5.36. PTEID_ReadFile

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: O caminho do ficheiro, por exemplo {0x3F, 0x00, 0x5F, 0x00, 0xEF, 0x02}, para o ficheiro ID.

filelen

in: Comprimento do ficheiro em *bytes*.

out

out: O *buffer* que guarda os conteúdos do ficheiro.

outlen

in/out: Número de *bytes* alocados / número de *bytes* lidos.

PinId

in: O identificador do PIN de Morada (apenas necessário aquando da leitura do ficheiro de Morada).

4.5.37. PTEID_ReadSOD

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.

4.5.38. PTEID_SelectADF

long PTEID_SelectADF(unsigned char *adf, long adflen)

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

Parâmetros

adf

in: A AID do ADF.

adflen

in: O comprimento do buffer que contém a AID.

4.5.39. PTEID_SendAPDU

long PTEID_SendAPDU(
const unsigned char *ucRequest,
unsigned long ulRequestLen,
unsigned char *ucResponse,
unsigned long *ulResponseLen)

Enviar uma APDU (Application Protocol Data Unit) para o cartão e receber a respetiva resposta.

Parâmetros:

ucRequest

in: APDU a enviar par o cartão.

ulRequestLen

in: Comprimento da APDU em bytes.

ucResponse

out: Espaço para a APDU de resposta.

ulResponseLen

in/out: Comprimento da APDU de resposta. A aplicação deve colocar o tamanho do *buffer* reservado para a resposta. A biblioteca colocará nesta variável o tamanho efetivo da resposta.

4.5.40. PTEID_SetChangeAddressCallback

void PTEID_SetChangeAddressCallback(
void(_USERENTRY * callback)(tAddressChangeState state))

Atribui ao *middleware* uma função de *callback*, definida pelo utilizador, para ser chamada durante a alteração do morada. Esta função é invocada pelo *middleware* de cada vez que o estado da operação se altera.

4.5.41. PTEID_SetSODCAs

long PTEID_SetSODCAs(PTEID_Certifs *Certifs)

Especificar os certificados (raiz) que são usados para assinar os certificados *Document Signer* 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 na diretoria *%APPDIR%/eidstore/certs* (*%APPDIR %* corresponde à diretoria onde a aplicação reside). Se esta diretoria não existir (ou não contiver os certificados corretos para o cartão), deverá chamar esta função para o especificar; ou desativar 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 uma estrutura *PTEID_Certifs*, ou NULL.

4.5.42. PTEID_SetSODChecking

long PTEID_SetSODChecking(int bDoCheck)

Ativar ou desativar 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 foi falsificada. Este processo é efetuado através da leitura do ficheiro SOD que contém *hashes* sobre os dados mencionados acima e está assinada por um certificado *Document Signer*.

Parâmetros:

bDoCheck

in: *true* para ativar verificação SOD, *false* para desativar.

4.5.43. PTEID_UnblockPIN

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 identificador do PIN, 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.

4.5.44. PTEID_UnblockPIN_Ext

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 identificador do PIN, ver o registo PTEID_Pins.

pszPuk
in: O valor do 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

4.5.45. PTEID_VerifyPIN

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

Verificar um PIN.

Parâmetros:

PinId

in: O identificador do PIN, 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.

4.5.46. PTEID_VerifyPIN_No_Alert

long PTEID_VerifyPIN_No_Alert(
unsigned char PinId,
char *Pin,
long *triesLeft)

Verificar um PIN. No caso de ser o PIN de assinatura, não é mostrada a mensagem de alerta e apenas é pedido o PIN ao utilizador. A utilizar quando se pretende simplesmente verificar o PIN, sem efetuar assinatura.

Parâmetros:

PinId

in: O identificador do PIN, 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.

4.5.47. PTEID_WriteFile

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 em bytes.

in

in: Os dados a ser escritos no ficheiro.

inlen

in: O comprimento dos dados a escrever.

PinId

in: O identificador do PIN de Autenticação, ver o registo PTEID_Pins.

4.5.48. PTEID_WriteFile_inOffset

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

Escrever dados para um ficheiro no cartão, especificando o índice a partir do qual se pretende escrever os dados.

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 em bytes.

in

in: Os dados a ser escritos no ficheiro.

inOffset

in: O *byte* a partir do qual se pretende escrever.

inlen

in: O comprimento dos dados a escrever.

PinId

in: O identificador do PIN de Autenticação, ver o registo PTEID_Pins.

4.6. *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:

- Alguns 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 o SOD que foi *cached* ainda está atualizado. 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 efetua um GetID() e depois GetAddress(), o SOD não é lido novamente para verificar a validade dos dados de Morada. Como exceção, quando CVC_Write_XXX() é chamado, é feito um *reset* à *flag*.

4.7. 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
```