

# Cartão de Cidadão

Versão 3.8.0

05/07/2022



# Tabela de Conteúdos

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Abreviaturas e acrónimos</b>	<b>4</b>
<b>3</b>	<b>Instalação</b>	<b>5</b>
3.1	Sistemas Operativos suportados . . . . .	5
3.2	Linguagens de programação . . . . .	5
3.3	Compiladores . . . . .	5
3.4	Instalação do SDK . . . . .	5
3.4.1	Windows . . . . .	5
3.4.2	Linux . . . . .	5
3.4.3	MacOS . . . . .	6
<b>4</b>	<b>Procedimentos</b>	<b>7</b>
4.1	Pré-condições . . . . .	7
4.2	Inicialização / Finalização do SDK . . . . .	7
4.3	Configurar modo teste . . . . .	8
4.4	Acesso ao <i>smartcard</i> Cartão de Cidadão . . . . .	8
4.4.1	Eventos de inserção / remoção de cartões . . . . .	10
4.5	Dados pessoais do cidadão . . . . .	11
4.5.1	Obtenção da Identificação . . . . .	11
4.5.2	Obtenção da fotografia . . . . .	12
4.5.3	Obtenção da morada . . . . .	13
4.5.4	Leitura e escrita das notas pessoais . . . . .	14
4.5.5	Leitura dos dados de identidade do Cidadão e da Morada . . . . .	15
4.5.6	Obtenção dos dados do cartão em formato XML . . . . .	17
4.6	PINs . . . . .	18
4.6.1	Verificação e alteração do PIN . . . . .	18
4.7	Assinatura Digital . . . . .	19
4.7.1	Formato XML Advanced Electronic Signatures (XAdES) . . . . .	20
4.7.2	Adicionar uma assinatura (XAdES) a um pacote ASiC . . . . .	21
4.7.3	Ficheiros PDF . . . . .	21
4.7.4	Assinatura de vários documentos PDF com uma única introdução de PIN . . . . .	23
4.7.5	Customização do selo de assinatura visível . . . . .	23
4.7.6	Configurar o servidor de selo temporal . . . . .	24
4.7.7	Bloco de dados . . . . .	25
4.8	Certificados digitais . . . . .	25
4.8.1	Leitura dos certificados digitais no cartão de cidadão ou da Chave Móvel Digital . . . . .	25
4.9	Sessão segura . . . . .	26
<b>5</b>	<b>Atualizações do Middleware</b>	<b>28</b>
<b>6</b>	<b>Tratamento de erros</b>	<b>29</b>
<b>7</b>	<b>API PKCS#11</b>	<b>30</b>
<b>8</b>	<b>Compatibilidade com o SDK da versão 1</b>	<b>32</b>
8.1	Métodos removidos . . . . .	32
8.2	Diferenças no comportamento de alguns métodos . . . . .	32
8.3	Códigos de Erro . . . . .	32
<b>9</b>	<b>Serviços online usados pelo Middleware</b>	<b>34</b>
<b>10</b>	<b>Notas do Utilizador</b>	<b>35</b>

# 1 Introdução

Este documento destina-se a programadores e analistas de sistemas que tencionam desenvolver soluções informáticas com base no SDK do middleware versão 3 do Cartão de Cidadão.

Esta versão do SDK disponibiliza a mesma interface (API) que a disponibilizada na versão 1 e 2 do SDK. Desta forma, pretende-se obter a retro-compatibilidade com versões anteriores do SDK. Embora a API anterior continue disponível, esta é desaconselhada pois a nova API cobre a maior parte dos casos de uso da anterior e tem funcionalidades novas.

A secção **Compatibilidade com o SDK da versão 1** contém informações relativamente à continuação da compatibilidade com a versão 1, descreve as diferenças comportamentais, os métodos removidos e os códigos de erro disponíveis.

O SDK do cartão de cidadão consiste num conjunto de bibliotecas utilizadas no acesso e suporte ao cartão de cidadão. Este SDK foi desenvolvido em C++, sendo providenciado o suporte a três diferentes tipos de sistemas operativos de 32/64 bits:

- Windows;
- Linux;
- MacOS;

Através dos exemplos presentes neste documento será possível desenvolver uma aplicação simples que interaja com o Cartão de Cidadão.

O desenvolvimento aplicacional utilizando o SDK pode ser realizado em C++ ou alternativamente em Java ou C# através de *wrappers* providenciados com o SDK.

Este manual serve de referência e ilustra alguns exemplos de utilização do SDK, para obter a documentação completa da API visite <https://github.com/amagovpt/autenticacao.gov> onde encontrará a documentação.

Para questões, sugestões ou comentários relativos à utilização do SDK, envie um e-mail para [info.cidadao@ama.pt](mailto:info.cidadao@ama.pt).

## 2 Abreviaturas e acrónimos

Acrónimos / abreviaturas	Definição
API	Application Programming Interface
SDK	Software Development Kit
PDF	Portable Document Format
XML	Extensible Markup Language
XAdES	XML Advanced Electronic Signatures
PAdES	PDF Advanced Electronic Signatures

## 3 Instalação

### 3.1 Sistemas Operativos suportados

A lista de sistemas operativos suportados, arquitecturas de 32 e 64 bits, são:

- Sistemas operativos Windows:
  - Windows 7;
  - Windows 8/8.1;
  - Windows 10;
  - Windows 11.
- Distribuições de Linux:
  - Ubuntu: 18.04 e superiores
  - OpenSuse: Leap 42.2
  - Fedora: 24 e superiores
- Sistemas operativos Apple MacOS:
  - MacOS Sierra (10.12) e superiores

### 3.2 Linguagens de programação

A lista de linguagens de programação suportadas são:

- C++: Em Windows, Linux e MacOS;
- Java: Em Windows, Linux e MacOS;
- C#: Apenas em Windows;

### 3.3 Compiladores

A lista de compiladores suportados são:

- C++:
  - Windows: Visual Studio 2017
  - Linux: GCC ou LLVM (clang);
  - MacOS: Compilador distribuído pela Apple. Dependendo da versão pode ser GCC ou LLVM (clang).
- Java - Oracle JDK 8 ou superior

### 3.4 Instalação do SDK

O SDK é distribuído nos pacotes de instalação Autenticação.gov que incluem também a aplicação GUI para o utilizador final. Os referidos pacotes de instalação são disponibilizados pela AMA em <https://www.autenticacao.gov.pt/web/guest/cc-aplicacao>

#### 3.4.1 Windows

Para instalar o SDK basta efectuar o *download* do ficheiro MSI de instalação e executar.

As bibliotecas C++ (pteidlibCpp.lib e respectivos *header files*), Java e C# ficarão disponíveis por defeito em C:\Program Files\PortugalIdentity Card\sdk\ ou na directoria seleccionada durante a instalação da aplicação, neste caso o SDK estará disponível em {directoria\_seleccionada}\sdk\ .

#### 3.4.2 Linux

Para instalar o SDK é necessario efectuar o *download* do pacote em formato deb ou rpm conforme a distribuição Linux que utiliza.

A biblioteca C++ libpteidlib.so ficará disponível em /usr/local/lib e os respectivos C++ *header files* em /usr/local/include

O SDK Java ficará disponível em `/usr/local/lib/pteid_jni`

Se a instalação for feita a partir do código fonte disponível em <https://github.com/amagovpt/autenticacao.gov> devem ser seguidas as instruções de compilação que constam do ficheiro README do projeto.

### 3.4.3 MacOS

Para instalar o SDK é necessário efectuar o *download* do pacote de instalação e executar.

O SDK Java ficará disponível em `/usr/local/lib/pteid_jni` . No que diz respeito ao SDK C++, os *header files* ficam localizados em `/usr/local/include` e a biblioteca à qual as aplicações deverão linkar está no caminho `/usr/local/lib/libpteidlib.dylib` .

## 4 Procedimentos

### 4.1 Pré-condições

#### 1. C/C++

- Windows/Visual Studio.
- Adicionar a import library **pteidlibCpp.lib** ao projecto.
- Por forma a conseguir incluir os *header files* do SDK adicionar a directoria C:\Program Files\Portugal Identity Card\sdk nas propriedades do projecto em "C/C++" → "General" → "Additional Include Directories".

#### 2. Java

- Incluir o ficheiro **pteidlibj.jar** como biblioteca no projecto.
- Em sistemas MacOS ou Linux, adicionar à propriedade de sistema `java.library.path` do projecto/aplicação Java a localização das bibliotecas nativas do SDK: `/usr/local/lib`
- As classes e métodos de compatibilidade com o middleware versão 1.x estão disponíveis no package **pteidlib** enquanto que as novas classes estão no *package* **pt.gov.cartaodecidadao**.

#### 3. C#

- Adicionar a biblioteca **pteidlib\_dotnet.dll** às *references* do projecto Visual Studio.
- As classes e métodos de compatibilidade estão no namespace **eidpt** enquanto que as novas classes estão no namespace **pt.portugal.eid**.

### 4.2 Inicialização / Finalização do SDK

A biblioteca **pteidlib** é inicializada através da invocação do método **PTEID\_initSDK()** (não é contudo obrigatório efectuar a inicialização). A finalização do SDK é obrigatória e deve ser efectuada através da invocação do método **PTEID\_releaseSDK()**, este método garante que todas as *threads* em segundo plano são terminadas e que a memória alocada para objectos internos do SDK é libertada.

#### 1. Exemplo em C++

```
1 #include "eidlib.h"
2
3 int main(int argc, char **argv){
4     PTEID_InitSDK();
5     // (...) - Código restante
6     PTEID_ReleaseSDK();
7 }
```

#### 2. Exemplo em Java

```
1 package pteidsample;
2 import pt.gov.cartaodecidadao.*;
3
4 /* NOTA: o bloco estático seguinte é estritamente necessário uma vez
5  que é preciso carregar explicitamente a biblioteca JNI que implementa
6  as funcionalidades do wrapper Java.*/
7
8 public class SamplePTEID {
9     static {
10         try {
11             System.loadLibrary("pteidlibj");
12         } catch (UnsatisfiedLinkError e) {
13             System.err.println("Native code library failed to load. \n" + e);
14             System.exit(1);
15         }
16     }
17 }
```

```
16     }
17
18     public static void main(String[] args) {
19         PTEID_ReaderSet.initSDK();
20         // (...) - Código restante
21         PTEID_ReaderSet.releaseSDK();
22     }
23 }
```

#### 1. Exemplo em C#

```
1 namespace PTEIDSample {
2     class Sample{
3         public static void Main(string[] args){
4             PTEID_ReaderSet.initSDK();
5             // (...) - Código restante
6             PTEID_ReaderSet.releaseSDK();
7         }
8     }
9 }
```

### 4.3 Configurar modo teste

Para alterar as configurações de forma a utilizar o modo teste, para usar cartões de teste, deve usar-se o método estático **SetTestMode**(*bool* bTestMode) da classe **PTEID\_Config**.

Com o valor do parâmetro *bTestMode* a *true*, os seguintes exemplos ativam o modo de teste.

#### 1. Exemplo C++

```
1 (...)
2 PTEID_Config::SetTestMode(true);
```

#### 1. Exemplo Java

```
1 (...)
2 PTEID_Config.SetTestMode(true);
```

#### 2. Exemplo C#

```
1 (...)
2 PTEID_Config.SetTestMode(true);
```

### 4.4 Acesso ao *smartcard* Cartão de Cidadão

Para aceder ao Cartão de Cidadão programaticamente devem ser efectuados os seguinte passos:

- Obter a lista de leitores de *smartcards* no sistema;



- Seleccionar um leitor de *smartcards*;
- Verificar se o leitor contém um cartão;
- Obter o objecto que fornece acesso ao cartão;
- Obter o objecto que contém os dados pretendidos;

A classe **PTEID\_ReaderSet** representa a lista de leitores de cartões disponíveis no sistema, esta classe disponibiliza uma variedade de métodos relativos aos leitores de cartões disponíveis. Através da lista de leitores, um leitor de cartões pode ser seleccionado resultando na criação de um objecto de contexto específico ao leitor em questão, a partir do qual é possível aceder ao cartão.

O objecto de contexto do leitor faculta o acesso ao cartão (se este estiver presente no leitor). O acesso ao cartão é obtido através do método **PTEID\_ReaderContext.getEIDCard()** que devolve um objecto do tipo **PTEID\_EIDCard**.

#### 1. Exemplo C++

```
1 PTEID_ReaderSet& readerSet = PTEID_ReaderSet::instance();
2 for( int i=0; i < readerSet.readerCount(); i++){
3     PTEID_ReaderContext& context = readerSet.getReaderByNum(i);
4     if (context.isCardPresent()){
5         PTEID_EIDCard &card = context.getEIDCard();
6         (...)
7     }
8 }
```

#### 1. Exemplo Java

```
1 PTEID_EIDCard card;
2 PTEID_ReaderContext context;
3 PTEID_ReaderSet readerSet;
4 readerSet = PTEID_ReaderSet.instance();
5 for( int i=0; i < readerSet.readerCount(); i++){
6     context = readerSet.getReaderByNum(i);
7     if (context.isCardPresent()){
8         card = context.getEIDCard();
9         (...)
10    }
11 }
```

#### 2. Exemplo C#

```
1 PTEID_EIDCard card;
2 PTEID_ReaderContext context;
3 PTEID_ReaderSet readerSet;
4 readerSet = PTEID_ReaderSet.instance();
5 for( int i=0; i < readerSet.readerCount(); i++){
6     context = readerSet.getReaderByNum(i);
7     if (context.isCardPresent()){
8         card = context.getEIDCard();
9         (...)
10    }
11 }
```

**Nota:** Uma forma rápida de obter um objecto de contexto será utilizar o método **getReader()**. Este método devolve o objecto de contexto do primeiro leitor com cartão que for encontrado no sistema. Alternativamente caso não existam cartões inseridos devolverá o primeiro leitor que encontrar no sistema.

- C++

```
PTEID_ReaderContext &readerContext = PTEID_ReaderSet.instance().getReader();
```

- Java

```
PTEID_ReaderContext readerContext = PTEID_ReaderSet.instance().getReader();
```

- C#

```
PTEID_ReaderContext readerContext = PTEID_ReaderSet.instance().getReader();
```

#### 4.4.1 Eventos de inserção / remoção de cartões

O SDK oferece uma forma de obter notificações dos eventos de cartão inserido e removido através de uma função *callback* definida pela aplicação. Para tal é necessário invocar o método **SetEventCallback()** no objecto **PTEID\_ReaderContext** associado ao leitor que se pretende monitorizar.

A função de *callback* definida deve ter a seguinte assinatura em C++:

```
void callback (long lRet, unsigned long ulState, void *callbackData)
```

O parâmetro *ulState* é a combinação de dois valores:

1. contador de eventos no leitor em causa
2. flag que indica se foi inserido ou removido um cartão

O parâmetro *lRet* é um código de erro que em caso de sucesso no acesso ao leitor terá sempre o valor 0.

O parâmetro *callbackData* é uma referência/ponteiro para o objecto que terá sido associado ao *callback* na função **SetEventCallback()**.

1. Exemplo Java:

```

1 class CardEventsCallback implements Callback {
2     @Override
3     public void getEvent(long lRet, long ulState, Object callbackData) {
4         int cardState = (int)ulState & 0x0000FFFF;
5         int eventCounter = ((int)ulState) >> 16;
6
7         System.err.println("DEBUG: Card Event:" +
8             " cardState: "+cardState + " Event Counter: "+ eventCounter);
9         if ((cardState & 0x0100) != 0)
10            {
11                System.out.println("Card inserted");
12            }
13         else {
14                System.out.println("Card removed");
15            }
16        }
17    }
18
19    PTEID_ReaderSet readerSet = PTEID_ReaderSet.instance();
20    PTEID_ReaderContext context = readerSet.getReader();
21    long callbackId = context.SetEventCallback(new CardEventsCallback(), null);
22    (...)
23    context.StopEventCallback(callbackId);

```

2. Exemplo C#:

```

1 public static void CardEventsCallback(int lRet, uint ulState, IntPtr
    callbackData) {
2
3     uint cardState = ulState & 0x0000FFFF;
4     uint eventCounter = (ulState) >> 16;
5
6     Console.WriteLine("DEBUG: Card Event: cardState: {1} Event Counter: {2}",
7                       cardState,
8                       eventCounter);
9
10    if ((cardState & 0x0100) != 0) {
11        Console.WriteLine("Card inserted");
12    }
13    else {
14        Console.WriteLine("Card removed");
15    }
16 }
17
18 PTEID_ReaderSet readerSet = PTEID_ReaderSet.instance();
19 IntPtr callbackData = (IntPtr)0;
20
21 PTEID_ReaderContext context = readerSet.getReader();
22 context.SetEventCallback(CardEventsCallback, callbackData);

```

## 4.5 Dados pessoais do cidadão

Os dados do cidadão e do cartão estão armazenados no cartão em múltiplos ficheiros. Destacam-se os seguintes ficheiros:

- ficheiro de identificação - contém os dados do cidadão/cartão impressos nas faces do cartão, incluindo a foto);
- ficheiro de morada – contém a morada do cidadão, este ficheiro é de acesso condicionado
- ficheiros de certificados do cidadão – contém os certificados de assinatura e autenticação do cidadão.
- ficheiros de certificados CA's.
- ficheiro de notas pessoais – é um ficheiro de leitura livre e de escrita condicionada onde o cidadão pode colocar até 1000 *bytes*.

### 4.5.1 Obtenção da Identificação

Para obter o conteúdo do ficheiro de identificação, o método **PTEID\_EIDCard.getID()** deverá ser utilizado.

#### 1. Exemplo C++

```

1 (...)
2 PTEID_EIDCard& card = context.getEIDCard();
3 PTEID_EId& eid = card.getID();
4
5 std::string nome = eid.getGivenName();
6 std::string nrCC = eid.getDocumentNumber();
7 (...)

```

#### 2. Exemplo Java

```

1 (...)

```

```
2 PTEID_EIDCard card = context.getEIDCard();
3 PTEID_EId eid = card.getID();
4
5 String nome = eid.getGivenName();
6 String nrCC = eid.getDocumentNumber();
7 (...)
```

### 3. Exemplo C#

```
1 (...)
2 PTEID_EIDCard card = context.getEIDCard();
3 PTEID_EId eid = card.getID();
4
5 string nome = eid.getGivenName();
6 string nrCC = eid.getDocumentNumber();
7 (...)
```

## 4.5.2 Obtenção da fotografia

A fotografia do cidadão está guardada no CC no formato JPEG2000 mas o SDK disponibiliza a fotografia no formato original e alternativamente em formato PNG.

### 1. Exemplo C++

```
1 (...)
2 PTEID_EIDCard& card = context.getEIDCard();
3 PTEID_EId& eid = card.getID();
4 PTEID_Photo& photoObj = eid.getPhotoObj();
5 PTEID_ByteArray& praw = photoObj.getphotoRAW(); // formato JPEG2000
6 PTEID_ByteArray& ppng = photoObj.getphoto();    // formato PNG
```

### 2. Exemplo Java

```
1 (...)
2 PTEID_EIDCard card = context.getEIDCard();
3 PTEID_EId eid = card.getID();
4 PTEID_Photo photoObj = eid.getPhotoObj();
5 PTEID_ByteArray praw = photoObj.getphotoRAW(); // formato JPEG2000
6 PTEID_ByteArray ppng = photoObj.getphoto();    // formato PNG
```

### 3. Exemplo C#

```
1 (...)
2 PTEID_EIDCard card = context.getEIDCard();
3 PTEID_EId eid = card.getID();
4 PTEID_Photo photoObj = eid.getPhotoObj();
5 PTEID_ByteArray praw = photoObj.getphotoRAW(); // formato JPEG2000
6 PTEID_ByteArray ppng = photoObj.getphoto();    // formato PNG
7 (...)
```

### 4.5.3 Obtenção da morada

O ficheiro da morada só pode ser lido após a verificação do pin da morada correcto.

Para mais informação sobre tratamento de erros gerados pelo SDK ver a secção **Tratamento de Erros**.

#### 1. Exemplo C++

```
1 // (...) - Inicialização
2
3 PTEID_EIDCard& eidCard = PTEID_ReaderSet::instance().getReader().getEIDCard();
4 unsigned long triesLeft;
5
6 PTEID_Pins& pins = eidCard.getPins();
7 PTEID_Pin& pin = pins.getPinByPinRef(PTEID_Pin::ADDR_PIN); //ADDR_PIN - Código
    de Morada
8
9 if (pin.verifyPin("", triesLeft, true)){
10     PTEID_Address& address = eidCard.getAddr();
11
12     std::cout << "Country:                " << address.getCountryCode()
13         << std::endl;
14     // (...) - Código restante
15     std::cout << "Postal Locality:        " << address.getPostalLocality
16         () << std::endl;
17 }
18
19 // (...) - Finalização
```

#### 2. Exemplo Java

```
1 // (...) - Inicialização
2
3 PTEID_EIDCard eidCard = PTEID_ReaderSet.instance().getReader().getEIDCard();
4 PTEID_ulwrapper triesLeft = new PTEID_ulwrapper(-1);
5
6 PTEID_Pins pins = eidCard.getPins();
7 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.ADDR_PIN); //ADDR_PIN - Código de
    Morada
8
9 if (pin.verifyPin("", triesLeft, true)){
10     PTEID_Address address = eidCard.getAddr();
11
12     System.out.println("Country:                " + address.
13         getCountryCode());
14     // (...) - Código restante
15     System.out.println("Postal Locality:        " + address.
16         getPostalLocality());
17 }
18
19 // (...) - Finalização
```

#### 3. Exemplo C#

```
1 // (...) - Inicialização
2
3 PTEID_EIDCard eidCard = PTEID_ReaderSet.instance().getReader().getEIDCard();
```

```

4 uint triesLeft = uint.MaxValue;
5
6 PTEID_Pins pins = eidCard.getPins();
7 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.ADDR_PIN); //ADDR_PIN - Código de
    Morada
8
9 if (pin.verifyPin("", ref triesLeft, true)){
10     PTEID_Address address = eidCard.getAddr();
11
12     System.out.println("Country:                " + address.
        getCountryCode());
13     // (...) - Código restante
14     System.out.println("Postal Locality:        " + address.
        getPostalLocality());
15 }
16
17 // (...) - Finalização

```

#### 4.5.4 Leitura e escrita das notas pessoais

Para ler as notas pessoais deverá ser utilizado o método **PTEID\_EIDCard.readPersonalNotes()**. Para a escrita de dados deverá ser utilizado o método **PTEID\_EIDCard.writePersonalNotes()**, sendo necessária a introdução do PIN de autenticação. Neste momento, as notas pessoais têm um limite de 1000 bytes (codificação recomendada: UTF-8).

##### 1. Exemplo C++

```

1 // (...) - Inicialização
2
3 PTEID_EIDCard& eidCard = PTEID_ReaderSet::instance().getReader().getEIDCard();
4
5 //Ler notas atuais e imprimir na consola
6 const char *my_notes = eidCard.readPersonalNotes();
7 std::cout << "Current notes: " << my_notes << std::endl;
8
9 //Escrever novas notas
10 std::string notes("We wrote successfully to the card!");
11 PTEID_ByteArray personalNotes((const unsigned char*) notes.c_str(), notes.size()
    + 1);
12 bool ok;
13
14 PTEID_Pins& pins = eidCard.getPins();
15 PTEID_Pin& pin = pins.getPinByPinRef(PTEID_Pin::AUTH_PIN); // AUTH_PIN - Código
    de Autenticação
16
17 ok = eidCard.writePersonalNotes(personalNotes, &pin);
18 std::cout << "Was writing successful? " << (ok ? "Yes!" : "No.") << std::endl;
19
20 // (...) - Finalização

```

##### 2. Exemplo Java

```

1 // (...) - Inicialização
2
3 PTEID_EIDCard eidCard = PTEID_ReaderSet.instance().getReader().getEIDCard();
4
5 //Ler notas atuais e imprimir na consola

```

```

6 String my_notes = eidCard.readPersonalNotes();
7 System.out.println("Current notes: " + my_notes);
8
9 //Escrever novas notas
10 String notes = "We wrote successfully to the card!";
11 PTEID_ByteArray personalNotes = new PTEID_ByteArray(notes.getBytes(), notes.
    getBytes().length);
12 boolean ok;
13
14 PTEID_Pins pins = eidCard.getPins();
15 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.AUTH_PIN); // AUTH_PIN - Código de
    Autenticação
16
17 ok = eidCard.writePersonalNotes(personalNotes, pin);
18 System.out.println("Was writing successful? " + (ok ? "Yes!" : "No."));
19
20 // (...) - Finalização

```

### 3. Exemplo C#

```

1 // (...) - Inicialização
2
3 PTEID_EIDCard eidCard = PTEID_ReaderSet.instance().getReader().getEIDCard();
4
5 //Ler notas atuais e imprimir na consola
6 String my_notes = eidCard.readPersonalNotes();
7 Console.WriteLine("Current notes: " + my_notes);
8
9 //Escrever novas notas
10 String notes = "We wrote successfully to the card!";
11
12 byte[] notesBytes = Encoding.UTF8.GetBytes(notes);
13 PTEID_ByteArray personalNotes = new PTEID_ByteArray(notesBytes, (uint)
    notesBytes.Length);
14 Boolean ok;
15
16 PTEID_Pins pins = eidCard.getPins();
17 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.AUTH_PIN); // AUTH_PIN - Código de
    Autenticação
18
19 ok = eidCard.writePersonalNotes(personalNotes, pin);
20 Console.WriteLine("Was writing successful? " + (ok ? "Yes!" : "No."));
21
22 // (...) - Finalização

```

#### 4.5.5 Leitura dos dados de identidade do Cidadão e da Morada

Para os métodos de prefixo "get" das classes **PTEID\_Eid** e **PTEID\_Address** não apresentamos exemplos já que estes métodos apenas são responsáveis pelas tarefas de obtenção dos campos específicos dentro dos ficheiros de identidade e morada e todos eles devolvem resultados do tipo **String** (no caso de Java/C#) ou **const char \*** (no caso da biblioteca C++)

##### **PTEID\_Eid**

Método	Descrição
getDocumentVersion()	versão do documento de identificação
getDocumentType()	tipo de documento - "Cartão de cidadão"

Método	Descrição
getCountry()	código do país no formato ISO3166
getGivenName()	nomes próprios do titular do cartão
getSurname()	apelidos do titular do cartão
getGender()	género do titular do cartão
getDateOfBirth()	data de nascimento
getNationality()	nacionalidade (código do país no formato ISO3166 )
getDocumentPAN()	número PAN do cartão (PAN - primary account number)
getValidityBeginDate()	data de emissão
getValidityEndDate()	data de validade
getLocalofRequest()	local de pedido do cartão
getHeight()	altura do titular do cartão
getDocumentNumber()	número do cartão de cidadão
getCivilianIdNumber()	número de identificação civil
getTaxNo()	número de identificação fiscal
getSocialSecurityNumber()	número de segurança social
getHealthNumber()	número de utente de saúde
getIssuingEntity()	entidade emissora do cartão
getGivenNameFather()	nomes próprios do pai do titular do cartão
getSurnameFather()	apelidos do pai do titular do cartão
getGivenNameMother()	nomes próprios da mãe do titular do cartão
getSurnameMother()	apelidos da mãe do titular do cartão
getParents()	filiação do titular do cartão no seguinte formato "nome e apelido do pai * nome e apelido da mãe"
getPhotoObj()	objecto que contém a foto do titular do cartão
getCardAuthKeyObj()	chave pública do cartão
getValidation()	indica se cartão se encontra válido
getMRZ1()	primeira linha do campo MRZ
getMRZ2()	segunda linha do campo MRZ
getMRZ3()	terceira linha do campo MRZ
getAccidentalIndications()	indicações eventuais

**PTEID\_\_Address**

Método	Descrição
getCountryCode()	código do país no formato ISO3166
getDistrict()	nome do distrito
getDistrictCode()	código do distrito
getMunicipality()	nome do município
getMunicipalityCode()	código do município
getCivilParish()	nome da freguesia
getCivilParishCode()	código da freguesia
getAbbrStreetType()	abreviatura do tipo de via
getStreetType()	tipo de via
getStreetName()	nome da via
getAbbrBuildingType()	abreviatura do tipo de edifício
getBuildingType()	tipo do edifício
getDoorNo()	número da entrada
getFloor()	número do piso
getSide()	lado
getLocality()	localidade
getPlace()	lugar
getZip4()	código postal
getZip3()	código postal
getPostalLocality()	localidade postal



**PTEID\_Address** - Apenas aplicável a moradas estrangeiras

Método	Descrição
getForeignCountry()	país
getForeignAddress()	endereço
getForeignCity()	cidade
getForeignRegion()	região
getForeignLocality()	localidade
getForeignPostalCode()	código postal

**PTEID\_Address** - Aplicável a ambas as moradas (nacionais e estrangeiras)

Método	Descrição
getGeneratedAddressCode()	código do endereço
isNationalAddress()	booleano que indica se é uma morada nacional

**4.5.6 Obtenção dos dados do cartão em formato XML**

Os dados do cidadão existentes no cartão podem ser extraídos em formato XML. A fotografia é retornada em base-64 no formato aberto PNG. Para além dos dados do cidadão é possível incluir também a área de notas pessoais.

## 1. Exemplo em C++

```

1 unsigned long triesLeft;
2 PTEID_EIDCard *card;
3 (...)
4 card->getPins().getPinByPinRef(PTEID_Pin::ADDR_PIN).verifyPin("", triesLeft,
    true);
5 PTEID_XmlUserRequestedInfo requestedInfo;
6 requestedInfo.add(XML_CIVIL_PARISH);
7 (...)
8 requestedInfo.add(XML_GENDER);
9 PTEID_CCXML_Doc &ccxml = card.getXmlCCDoc(requestedInfo);
10 const char * resultXml = ccxml.getCCXML();

```

## 2. Exemplo em Java

```

1 String resultXml;
2 PTEID_EIDCard card;
3 PTEID_ulwrapper triesLeft = new PTEID_ulwrapper(-1);
4 (...)
5 card.getPins().getPinByPinRef(PTEID_Pin.ADDR_PIN).verifyPin("", triesLeft, true)
    ;
6 PTEID_XmlUserRequestedInfo requestedInfo = new PTEID_XmlUserRequestedInfo();
7 requestedInfo.add(XMLUserData.XML_CIVIL_PARISH);
8 (...)
9 requestedInfo.add(XMLUserData.XML_GENDER);
10 PTEID_CCXML_Doc result = idCard.getXmlCCDoc(requestedInfo);
11 resultXml = result.getCCXML();

```

## 3. Exemplo em C#

```

1 string resultXml;
2 PTEID_EIDCard card;
3 uint triesLeft;
4 (...)
5 card.getPins().getPinByPinRef(PTEID_Pin.ADDR_PIN).verifyPin("", ref triesLeft,
    true);
6 PTEID_XmlUserRequestedInfo requestedInfo = new PTEID_XmlUserRequestedInfo();
7 requestedInfo.add(XMLUserData.XML_CIVIL_PARISH);
8 (...)
9 requestedInfo.add(XMLUserData.XML_GENDER);
10 PTEID_CCXML_Doc result = idCard.getXmlCCDoc(requestedInfo);
11 resultXml = result.getCCXML();

```

## 4.6 PINs

### 4.6.1 Verificação e alteração do PIN

Para verificação do PIN deverá ser utilizado o método **verifyPin()**. Para a sua alteração, deverá ser utilizado o método **changePin()**.

#### 1. Exemplo C++

```

1 PTEID_EIDCard& card;
2 unsigned long triesLeft;
3 (...)
4 PTEID_Pins &pins = card.getPins();
5 PTEID_Pin &pin = pins.getPinByPinRef(PTEID_Pin.ADDR_PIN);
6 if (pin.verifyPin("", &triesLeft, true)){
7     bool bResult = pin.changePin("", "", triesLeft, pin.getLabel());
8     if (!bResult && -1 == triesLeft) return;
9 }

```

#### 2. Exemplo Java

```

1 PTEID_EIDCard card;
2 PTEID_ulwrapper triesLeft = new PTEID_ulwrapper(-1);
3 (...)
4 PTEID_Pins pins = card.getPins();
5 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.ADDR_PIN);
6 if (pin.verifyPin("", triesLeft, true)){
7     bool bResult = pin.changePin("", "", triesLeft, pin.getLabel());
8     if (!bResult && -1 == triesLeft) return;
9 }

```

#### 3. Exemplo C#

```

1 PTEID_EIDCard card;
2 uint triesLeft;
3 (...)
4 PTEID_Pins pins = card.getPins();
5 PTEID_Pin pin = pins.getPinByPinRef(PTEID_Pin.ADDR_PIN);
6 if (pin.verifyPin("", ref triesLeft, true)){

```

```
7     bool bResult = pin.changePin("", "", triesLeft, pin.getLabel());
8     if (!bResult && -1 == triesLeft) return;
9 }
```

**Nota:** Se o primeiro parâmetro do método `verifyPin` for a string vazia, será aberta uma janela para introdução do PIN. Caso contrário, o primeiro parâmetro deverá ser a string com o PIN a ser verificado. Esta lógica aplica-se de modo análogo aos dois primeiros argumentos do método `changePin`.

## 4.7 Assinatura Digital

A funcionalidade de assinatura está disponível para classes que implementam `PTEID_SigningDevice` como `PTEID_EIDCard` e `PTEID_CMDSignatureClient`.

Ao assinar com `PTEID_EIDCard` será apresentada uma janela para introdução do PIN de assinatura do cartão inserido no leitor.

Com `PTEID_CMDSignatureClient`, é apresentada uma janela para introdução do número de telemóvel e PIN associados à conta da Chave Móvel Digital do assinante seguido de uma janela para introdução do código de segurança enviado para o telemóvel. Antes de utilizar os serviços da CMD deve configurar as credenciais de acesso ao serviço através do método `setCredentials`. Para mais informação, deverá contactar a AMA pelo email [info.cidadao@ama.pt](mailto:info.cidadao@ama.pt).

Para obter um `PTEID_SigningDevice` deve utilizar a classe `PTEID_SigningDeviceFactory`. Na chamada ao método `getSigningDevice` pode indicar qual dos tipos de `PTEID_SigningDevice` deseja obter. Se ativar múltiplas opções, será apresentada uma janela ao utilizador para escolher o método de assinatura: Cartão de Cidadão ou Chave Móvel Digital.

**NOTA:** As classes `PTEID_CMDSignatureClient`, `PTEID_SigningDevice` e `PTEID_SigningDeviceFactory` estão disponíveis a partir da versão 3.8.0 do Middleware. Em versões anteriores existiam apenas os métodos de assinatura da classe `PTEID_EIDCard` para assinar com Cartão de Cidadão.

### 1. Exemplo C++

```
1 (...)
2 // Necessário para chamadas com PTEID_CMDSignatureClient
3 PTEID_CMDSignatureClient::setCredentials(BASIC_AUTH_USER, BASIC_AUTH_PASSWORD,
4     BASIC_AUTH_APPID);
5 (...)
6 // Neste exemplo ambas as opções estão ativas e, por isso, será apresentada uma
7 // janela
8 PTEID_SigningDeviceFactory &factory = PTEID_SigningDeviceFactory::instance();
9 PTEID_SigningDevice &signingDev = factory.getSigningDevice(true, true);
10 (...)
```

### 2. Exemplo Java

```
1 (...)
2 // Necessário para chamadas com PTEID_CMDSignatureClient
3 PTEID_CMDSignatureClient.setCredentials(BASIC_AUTH_USER, BASIC_AUTH_PASSWORD,
4     BASIC_AUTH_APPID);
5 (...)
6 // Neste exemplo ambas as opções estão ativas e, por isso, será apresentada uma
7 // janela
8 PTEID_SigningDeviceFactory factory = PTEID_SigningDeviceFactory.instance();
9 PTEID_SigningDevice signingDev = factory.getSigningDevice(true, true);
10 (...)
```

### 3. Exemplo C#

```

1 (...)
2 // Necessário para chamadas com PTEID_CMDSignatureClient
3 PTEID_CMDSignatureClient.setCredentials(BASIC_AUTH_USER, BASIC_AUTH_PASSWORD,
    BASIC_AUTH_APPID);
4 (...)
5 // Neste exemplo ambas as opções estão ativas e, por isso, será apresentada uma
    janela
6 PTEID_SigningDeviceFactory factory = PTEID_SigningDeviceFactory.instance();
7 PTEID_SigningDevice signingDev = factory.getSigningDevice(true, true);
8 (...)

```

#### 4.7.1 Formato XML Advanced Electronic Signatures (XAdES)

Esta funcionalidade permite assinar um ou múltiplos ficheiros em qualquer formato utilizando ou não selos temporais.

O método **SignXades** produz um ficheiro .zip que contém os ficheiros assinados e um ficheiro XML com a assinatura. O formato deste ficheiro .zip segue a [norma europeia ASIC](#) para *containers* de assinatura.

Será apresentado apenas um exemplo C++ para esta funcionalidade embora os wrappers Java e C# contêm exactamente as mesmas classes e métodos necessários **PTEID\_SigningDevice.SignXades()**.

##### 1. Exemplo C++

```

1 const char *ficheiros[] = {"teste/Ficheiro1", "teste/Ficheiro2", "teste/
    Ficheiro3"};
2 const char *destino = "teste/ficheiros_assinados.asice";
3 int n_paths = 3; // tamanho do array de ficheiros
4
5 /* perfil da assinatura:
6 PTEID_LEVEL_BASIC (XAdES-B), PTEID_LEVEL_TIMESTAMP (XAdES-T), PTEID_LEVEL_LTV (
    XAdES-LTA) */
7 PTEID_SignatureLevel level = PTEID_LEVEL_BASIC;
8
9 // assinar (1 única assinatura para todos os ficheiros)
10 signingDev.SignXades(destino, ficheiros, n_paths, level);
11 (...)

```

**Nota:** Alternativamente é possível assinar individualmente cada ficheiro da seguinte forma:

- Sem selo temporal (XAdES-B):
  - C++
 

```
card.SignXadesIndividual( dirDestino, ficheiros, n_paths );
```
  - Java/C#
 

```
card.SignXadesIndividual( dirDestino, ficheiros, ficheiros.length );
```
- Com selo temporal (XAdES-T):
  - C++
 

```
card.SignXadesTIndividual( dirDestino, ficheiros, n_paths );
```
  - Java/C#
 

```
card.SignXadesTIndividual( dirDestino, ficheiros, ficheiros.length );
```

- Para arquivo de longo período temporal (XAdES-LTA):
  - C++

```
card.SignXadesAIndividual( dirDestino, ficheiros, n_paths );
```
  - Java/C#

```
card.SignXadesAIndividual( dirDestino, ficheiros, ficheiros.length );
```

O parâmetro **dirDestino** contém a directoria destino onde serão colocados os ficheiros assinados.

**Nota 2:** Se for emitida a exceção com código `EIDMW_TIMESTAMP_ERROR` durante uma assinatura XAdES-T ou XAdES-LTA, significa que a aplicação do *timestamp* na assinatura falhou ou, no caso dos métodos de assinatura individual, que falhou para pelo menos uma das assinaturas. Neste caso, as assinaturas cujo *timestamping* falhou ficam com nível XAdES-B.

**Nota 3:** De modo semelhante à nota anterior, se for emitida a exceção com código `EIDMW_LTV_ERROR` numa assinatura XAdES-LTA, então a aplicação do *timestamp* sobre os dados de revogação não foi corretamente adicionado. Nesse caso, as assinaturas cujo *timestamping* falhou ficam com nível XAdES-T ou XAdES-LT.

#### 4.7.2 Adicionar uma assinatura (XAdES) a um pacote ASiC

Esta funcionalidade permite adicionar uma assinatura XAdES a um pacote ASiC existente, utilizando ou não selos temporais. Esta assinatura cobre todos os ficheiros de *input* incluídos no pacote ASiC.

O método **SignASiC** adiciona um ficheiro XML com a assinatura (XAdES-B/XAdES-T/XAdES-LTA), a um pacote ASiC existente, cujo formato siga a [norma europeia ASiC](#) para *containers* de assinatura. Este método recebe como primeiro parâmetro o caminho para o pacote ASiC. Como segundo parâmetro o nível/perfil da assinatura a incluir: XAdES-B (`PTEID_LEVEL_BASIC`), XAdES-T (`PTEID_LEVEL_TIMESTAMP`), XAdES-LTA (`PTEID_LEVEL_LTV`).

Será apresentado apenas um exemplo C++ para esta funcionalidade embora os wrappers Java e C# contêm exactamente as mesmas classes e métodos necessários **PTEID\_SigningDevice.SignASiC()**.

##### 1. Exemplo C++

```
1 const char *container = "teste/exemplo.asics";
2 PTEID_SignatureLevel level = PTEID_LEVEL_BASIC;
3
4 signingDev.SignASiC(container, level);
5 (...)
```

#### 4.7.3 Ficheiros PDF

O SDK fornece métodos para assinatura de ficheiros PDF de acordo com os standards PAdES (ETSI TS 102 778-1) e com o standard mais antigo implementado pelo Adobe Reader e Acrobat (*ISO 32000*).

As assinaturas produzidas pelas bibliotecas do SDK podem ser validadas com os referidos produtos da Adobe ou alternativas *opensource* como a biblioteca iText (<http://itextpdf.com>).

Os métodos de assinatura de PDF fornecem as seguintes opções:

- Assinatura de acordo com a especificação dos seguintes perfis:
  - PAdES-B: O nível mais simples para assinaturas de validade limitada à data de validade do certificado de assinatura do CC (até 5 anos ou até 10 anos). Este perfil não inclui selo temporal.
  - PAdES-T: Inclui um *timestamp* que prova o momento em que foi realizada a assinatura.
  - PAdES-LT: Para além dos requisitos do nível PAdES-T, são adicionados os dados necessários para validar o certificado usado para a assinatura digital (respostas OCSP e CRLs).

- PAdES-LTA: Este nível é recomendado a documentos que estão destinados a serem arquivados por um longo período de tempo. Inclui os requisitos do nível PAdES-LT e, adicionalmente, um *timestamp* que garante a integridade dos dados de validação. Deste modo, é possível provar no futuro que no momento da assinatura o certificado do cartão e respectiva cadeia não estavam revogados ou expirados.
- Assinatura de vários ficheiros em *batch* (com apenas uma introdução de PIN).
- Inclusão de detalhes adicionais como a localização ou motivo da assinatura.
- Personalização do aspecto da assinatura no documento (página, localização na mesma e tamanho da assinatura).

A localização da assinatura, na página do documento a assinar, é definida a partir do canto superior esquerdo do rectângulo de assinatura através de coordenadas (x,y) expressas em percentagem da largura/altura da página em que o ponto (0,0) se situa no canto superior esquerdo da página. De notar que usando este método existem localizações que produzem uma assinatura truncada na página já que o método de assinatura não valida se a localização é válida para o "selo de assinatura" a apresentar.

Será apresentado apenas um exemplo C++ para esta funcionalidade embora os wrappers Java e C# contêm exactamente as mesmas classes e métodos necessários **PTEID\_PdfSignature()** e **PTEID\_SigningDevice.SignPDF()**.

Exemplo C++:

```
1 #include "eidlib.h"
2 (...)
3 //Ficheiro PDF a assinar
4 PTEID_PdfSignature signature("/home/user/input.pdf");
5
6 /* Adicionar uma imagem customizada à assinatura visível
7    0 array de bytes image_data deve conter uma imagem em formato
8    JPEG com as dimensões recomendadas (351x77 px) */
9 PTEID_ByteArray jpeg_data(image_data, image_length);
10 signature.setCustomImage(jpeg_data);
11
12 // No caso de se querer o formato pequeno da assinatura
13 signature.enableSmallSignatureFormat();
14
15 /* Configurar o perfil da assinatura:
16 PAdES-B: PTEID_SignatureLevel::PTEID_LEVEL_BASIC (configurado por defeito)
17 PAdES-T: PTEID_SignatureLevel::PTEID_LEVEL_TIMESTAMP
18 PAdES-LT: PTEID_SignatureLevel::PTEID_LEVEL_LT
19 PAdES-LTA: PTEID_SignatureLevel::PTEID_LEVEL_LTV */
20 signature.setSignatureLevel(PTEID_SignatureLevel::PTEID_LEVEL_TIMESTAMP);
21
22 //Especificar local da assinatura e motivo
23 const char * location = "Lisboa, Portugal";
24 const char * reason = "Concordo com o conteúdo do documento";
25
26 //Especificar o número da página e a posição nessa mesma página onde a
27   indicação visual da assinatura aparece
28 int page = 1;
29 double pos_x = 0.1; //Valores de 0 a 1
30 double pos_y = 0.1; //Valores de 0 a 1
31 eidCard.SignPDF(signature, page, pos_x, pos_y, location, reason, "/home/user/
   output.pdf");
```

**Nota:** Se for emitida a exceção com código **EIDMW\_TIMESTAMP\_ERROR** durante uma assinatura PAdES-T, PAdES-LT ou PAdES-LTA, significa que a aplicação do *timestamp* em uma ou mais assinaturas falhou. Neste caso, as assinaturas cujo *timestamping* falhou ficam com nível PAdES-B.

**Nota 2:** De modo semelhante à nota anterior, se for emitida a exceção com código **EIDMW\_LTV\_ERROR** numa

assinatura PAdES-LT ou PAdES-LTA, significa que não foi possível adicionar os dados de revogação ou o selo temporal sobre esses dados. Nesse caso, as assinaturas cujo *timestamping* falhou ficam com nível PAdES-T ou PAdES-LT dependendo se os dados de revogação foram corretamente adicionados.

#### 4.7.4 Assinatura de vários documentos PDF com uma única introdução de PIN

Esta funcionalidade permite assinar vários documentos PDF introduzindo o PIN somente uma vez. O selo visual de assinatura será aplicado na mesma página e localização em todos os documentos, sendo que a localização é especificada tal como na assinatura simples. Deverá ser utilizado o método **addToBatchSigning()** para construir a lista de documentos a assinar.

Será apresentado apenas um exemplo C++ para esta funcionalidade embora os wrappers Java e C# contenham exactamente as mesmas classes e métodos necessários na classe **PTEID\_PDFSignature()**.

Exemplo C++

```
1 #include "eidlib.h"
2 (...)
3 PTEID_PDFSignature signature;
4
5 //Para realizar uma assinatura em batch adicionar todos os ficheiros usando o
   seguinte método antes de invocar o signingDev.SignPDF()
6 signature.addToBatchSigning("File_to_Sign.pdf");
7 signature.addToBatchSigning("Other_File.pdf");
8 (...)
9
10 //Especificar local da assinatura e motivo
11 const char * location = "Lisboa, Portugal";
12 const char * reason = "Concordo com o conteúdo do documento";
13
14 //Especificar o número da página e a posição nessa mesma página onde a
   indicação visual da assinatura aparece
15 int page = 1;
16 double pos_x = 0.1; //Valores de 0 a 1
17 double pos_y = 0.1; //Valores de 0 a 1
18
19 //Para uma assinatura em batch, este parâmetro aponta para a directoria de
   destino
20 const char * output_folder = "/home/user/signed-documents/";
21 signingDev.SignPDF(signature, page, pos_x, pos_y, location, reason,
   output_folder);
22 (...)
```

#### 4.7.5 Customização do selo de assinatura visível

Através do método **setCustomSealSize(width, height)**, é possível alterar as dimensões do selo de assinatura visível utilizando o SDK. Para tal, é só necessário chamar o método com as dimensões pretendidas, após inicializar uma instância de **PTEID\_PDFSignature**. Este método pode ser chamado para a assinatura de um único ficheiro ou para multi-assinatura. Exemplo C++

```
1 // (...) - Inicialização
2
3 //Inicializar assinatura
4 PTEID_PDFSignature signature;
5
6 // (...) - Adicionar ficheiros e outras configurações (motivo, localização, etc)
7
8
```

```
9 //Para definir as dimensões do selo de assinatura é necessário invocar o
   seguinte método com as dimensões pretendidas.
10 signature.setCustomSealSize(200, 200);
11
12 // (...) - Assinar e Finalização
```

#### Exemplo Java

```
1 // (...) - Inicialização
2
3 PTEID_PDFSignature signature = new PTEID_PDFSignature();
4
5 // (...) - Adicionar ficheiros e outras configurações (motivo, localização, etc)
6
7 //Para definir as dimensões do selo de assinatura é necessário invocar o
   seguinte método com as dimensões pretendidas.
8 signature.setCustomSealSize(200, 200);
9
10 // (...) - Assinar e Finalização
```

#### Exemplo C#

```
1 // (...) - Inicialização
2
3 PTEID_PDFSignature signature = new PTEID_PDFSignature();
4
5 // (...) - Adicionar ficheiros e outras configurações (motivo, localização, etc)
6
7
8 //Para definir as dimensões do selo de assinatura é necessário invocar o
   seguinte método com as dimensões pretendidas.
9 signature.setCustomSealSize(200, 200);
10
11 // (...) - Assinar e Finalização
```

É também possível escolher incluir, ou não, a data de assinatura e o número de identificação civil através da aplicação, no menu **Personalização de Assinatura**, seleccionando as respetivas checkboxes, ou através do ficheiro `~/config/pteid.conf`, modificando o valor de `signature_seal_options`. Estas alterações são depois refletidas no selo da assinatura através do SDK.

#### 4.7.6 Configurar o servidor de selo temporal

O SDK permite seleccionar uma servidor diferente para a obtenção de selos temporais, uma vez que o servidor por defeito do Cartão do Cidadão (<http://ts.cartaodecidadao.pt/tsa/server>) tem um limite máximo de 20 pedidos em cada período de 20 minutos que se podem efectuar. Se este valor for excedido o serviço será bloqueado durante 24 horas, sem prejuízo de outras consequências em caso de repetição de situações de bloqueio. (para mais informações sobre o serviço de selo temporal/timestamps do Cartão do Cidadão, consulte a página <https://pki.cartaodecidadao.pt>).

Para usar um servidor diferente basta criar uma nova configuração, da seguinte forma:

```
1 PTEID_Config config = new PTEID_Config(PTEID_PARAM_XSIGN_TSAURL);
2 config.setString("http://sha256timestamp.ws.symantec.com/sha256/timestamp");
```



Após esta configuração tanto as assinaturas de documentos PDF (PAdES) bem como a assinaturas em formato XAdES vão usar este novo servidor configurado para obter os selos temporais ao assinar.

#### 4.7.7 Bloco de dados

Esta funcionalidade permite assinar um bloco de dados usando ou não o certificado de assinatura.

Para isso deverá ser utilizado o método **Sign()** numa classe que implemente **PTEID\_SigningDevice**.

O Algoritmo de assinatura suportado é o **RSA-SHA256** mas o *smartcard* apenas implementa o algoritmo RSA e como tal o bloco de input deve ser o *hash* **SHA-256** dos dados que se pretende assinar.

##### 1. Exemplo C++

```
1 PTEID_ByteArray data_to_sign;  
2 (...)  
3 PTEID_ByteArray output = signingDev.Sign(data_to_sign, true);  
4 (...)
```

##### 2. Exemplo Java

```
1 PTEID_ByteArray data_to_sign;  
2 (...)  
3 PTEID_ByteArray output= signingDev.Sign(data_to_sign, true);  
4 (...)
```

##### 3. Exemplo C#

```
1 PTEID_ByteArray data_to_sign, output;  
2 (...)  
3 PTEID_ByteArray output;  
4 output = signingDev.Sign(data_to_sign, true);  
5 (...)
```

## 4.8 Certificados digitais

### 4.8.1 Leitura dos certificados digitais no cartão de cidadão ou da Chave Móvel Digital

Os métodos para leitura dos certificados estão expostos nas classes que implementam **PTEID\_SigningDevice**.

Os métodos **getRoot()**, **getCA()**, **getSignature()** e **getAuthentication()** estão descontinuados. Para obter os certificados sugere-se invocar o método **getCertificates()** e obter o certificado/construir a cadeia manualmente.

Para **PTEID\_EIDCard**, o método **getCertificates()** devolve uma instância **PTEID\_Certificates** que contém os certificados do cartão inserido no leitor e das CAs necessárias à construção da cadeia completa.

Para **PTEID\_CMDSignatureClient**, o método **getCertificates()** devolve uma instância **PTEID\_Certificates** com a cadeia de certificados usada na última assinatura com essa instância de **PTEID\_CMDSignatureClient**. Se nenhuma assinatura tiver sido efetuada, é mostrada uma janela para autenticar com a conta da Chave

Móvel Digital para a qual se desejam obter os certificados. Este método requer credenciais de acesso ao serviço CMD tal como os métodos de assinatura (ver secção Assinatura Digital).

#### 1. Exemplo C++

```
1 (...)
2 PTEID_Certificates &certificates = signingDev.getCertificates();
```

#### 2. Exemplo Java/C#

```
1 (...)
2 PTEID_Certificates certs = card.getCertificates();
```

## 4.9 Sessão segura

O Cartão de Cidadão permite o estabelecimento de sessões seguras. É efectuada a autenticação entre ambas as partes (a aplicação e o cartão). Após este processo as operações seguintes são efectuadas sobre comunicação cifrada e autenticada.

A autenticação da aplicação é efectuada através de CVCs (Card Verifiable Certificates). Estes certificados são emitidos somente a entidades que estejam autorizadas em Lei a efectuar operações privilegiadas no cartão.

Existem duas operações privilegiadas que obrigam ao estabelecimento prévio de uma sessão segura:

1. Leitura da morada sem introdução de PIN.
2. Alteração da morada.

Exemplo em C para a leitura da morada sem introdução do PIN (utilizando a biblioteca OpenSSL para implementar a assinatura do desafio enviado pelo cartão).

Foram omitidos do bloco de código seguinte as declarações de *#include* necessárias para utilizar as funções do OpenSSL. Para mais informações sobre OpenSSL, consultar a wiki do projecto em: <https://wiki.openssl.org>.

Esta funcionalidade está apenas disponível nos métodos de compatibilidade com a versão 1 do Middleware.

Em seguida é apresentado o exemplo em C mas a sequência de métodos do SDK a utilizar em Java ou C# será a mesma, isto é:

1. **pteid.CVC\_Init()**
2. **pteid.CVC\_Authenticate()**
3. **pteid.CVC\_ReadFile()** ou **pteid.CVC\_GetAddr()**

```
1 //Função auxiliar para carregar a chave privada associada ao certificado CVC
2 RSA * loadPrivateKey(char * file_path) {
3     FILE * fp = fopen(file_path, "r");
4     if (fp == NULL) {
5         fprintf(stderr, "Failed to open private key file: %s!\n", file_path);
6         return NULL;
7     }
8     RSA * key = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
9     if (key == NULL) {
10        fprintf(stderr, "Failed to load private key file!\n");
11    }
12    return key;
```

```
13 }
14
15 //Init OpenSSL
16 OpenSSL_add_all_algorithms();
17 ERR_load_crypto_strings();
18 (...)
19 unsigned char challenge[128];
20 // challenge that was signed by the private key corresponding to the CVC
21 unsigned char signature[128];
22 unsigned char fileBuffer[2000];
23 long ret;
24 ret = PTEID_CVC_Init( cvcCert, cvcCert_len, challenge, sizeof(challenge));
25 if ( ret != 0 ){
26     PTEID_Exit(0);
27     return 1;
28 }
29 // private_key_path - path for private key file in
30 RSA* rsa_key = loadPrivateKey(private_key_path);
31 RSA_private_encrypt( sizeof(challenge), challenge, signature, rsa_key,
32     RSA_NO_PADDING);
33 ret = PTEID_CVC_Authenticate( signature, sizeof(signature) );
34 if ( ret != 0 ){
35     PTEID_Exit(0);
36     return 1;
37 }
38 unsigned char fileID[] = { /* address for file */ };
39 unsigned long outlen = sizeof(fileBuffer);
40 ret = PTEID_CVC_ReadFile( fileID, sizeof(fileID), fileBuffer, &outlen );
41 if ( ret != 0 ){
42     PTEID_Exit(0);
43     return 1;
44 }
45 (...)
46 PTEID_ADDR addrData; //For CVC_GetAddr()
47 ret = PTEID_CVC_GetAddr( &addrData );
48 if ( ret != 0 ){
49     PTEID_Exit(0);
50     return 1;
51 }
```

## 5 Atualizações do Middleware

Os SDKs Java e .Net estão dependentes de bibliotecas nativas que são alteradas em cada nova versão do middleware. Não é garantida a compatibilidade entre diferentes versões dos componentes Java/.Net e nativos.

A garantia de retro-compatibilidade deste projeto é apenas a nível da API Java ou .Net disponibilizada.

A recomendação que fazemos é que as aplicações Java que utilizem o SDK deverão carregar o `pteidlibj.jar` a partir da versão instalada pelo Middleware em vez de incluir a versão inicial que existia no momento do desenvolvimento.

Para o **SDK Java** é apenas necessário adicionar à *classpath* da aplicação o caminho para o `pteidlibj.jar` que é indicado na secção **Instalação do SDK** e garantir que na instalação da aplicação não é incluída outra versão do mesmo JAR.

Para o **SDK .Net** recomendamos o seguinte método para fazer uma atualização do Middleware e garantir a compatibilidade com as aplicações:

Cenário: Foi adicionada uma referência para a versão X da DLL `pteidlib_dotnet` na aplicação e pretende-se atualizar o middleware para a versão X + Y. É possível fazê-lo sem alteração ou recompilação do projeto através do ficheiro de configuração da aplicação usando um *assembly binding redirect*. Por exemplo, para atualizar para a versão 3.7.0 suportando uma aplicação desenvolvida com a 3.1.2, adicionar no ficheiro `Program.exe.config` o seguinte bloco:

```
1 <runtime>
2   <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
3     <dependentAssembly>
4       <assemblyIdentity name="pteidlib_dotnet"
5                           publicKeyToken="1fa91d379e36932f"
6                           culture="neutral" />
7       <bindingRedirect oldVersion="3.1.2.3831" newVersion="3.7.0.4572"/>
8       <codeBase version="3.7.0.4572" href="file:\\c:\Program Files\Portugal
9         Identity Card\sdk\dotnet\pteidlib_dotnet.dll" />
10     </dependentAssembly>
11   </assemblyBinding>
12 </runtime>
```

**Nota:** O procedimento descrito anteriormente só é possível se a `pteidlib_dotnet.dll` original era um *strong-named assembly*. Isto pode ser confirmado verificando a identidade de uma determinada DLL com o comando `sn.exe -T pteidlib_dotnet.dll`. Deverá ser retornado o valor indicado no bloco de código acima: `1fa91d379e36932f`. As versões da `pteidlib_dotnet.dll` anteriores à 3.0.15 (Setembro de 2018) não eram *strong-named assemblies* e como tal não podem ser atualizadas por este método, ou seja, exigem uma recompilação da aplicação com a referida DLL atualizada.

Referência: [SN.exe - Microsoft docs](#)

## 6 Tratamento de erros

O SDK do middleware trata os erros através do lançamento de excepções qualquer que seja a linguagem utilizada: C++, Java ou C#.

Os métodos de compatibilidade com a versão 1 do Middleware (MW) usam outros mecanismos de tratamento de erros: para mais detalhes consultar a secção **Códigos de Erro**.

A classe base de todas as excepções do MW é a classe **PTEID\_Exception**.

Existem algumas subclasses de **PTEID\_Exception** para erros específicos como **PTEID\_ExNoCardPresent** ou **PTEID\_ExCardBadType**.

Em todos os casos é sempre possível obter um código de erro numérico para todos os erros que estão tipificados nos métodos do MW através da chamada ao método **GetError()** da classe **PTEID\_Exception**. Através do método **GetMessage()** é possível obter uma descrição legível do erro (em língua inglesa)

As constantes numéricas dos códigos de erro estão expostas às aplicações em:

- C++: no ficheiro de include **eidErrors.h**
- C#: membros públicos da classe **pteidlib\_dotNet** com o prefixo **EIDMW**
- Java: membros públicos da interface **pteidlib\_JavaWrapperConstants** com o prefixo **EIDMW**

## 7 API PKCS#11

É possível o acesso aos certificados e operações associadas do CC através de uma API standard e multi-plataforma para dispositivos criptográficos que está descrita na norma [PKCS#11](#).

Aplicações que possam utilizar vários tipos de certificados e/ou dispositivos de autenticação/assinatura podem deste modo aceder às operações disponibilizadas pelo CC através de uma API comum se tiverem em conta as especificidades que indicamos em seguida.

Por exemplo em aplicações Java é possível utilizar o módulo pteid-pkcs11 incluído no middleware do CC através do *Security Provider* "SunPKCS11" da seguinte forma:

```
1      /* Exemplo de carregamento do "token" PKCS#11 que permite aceder às
2         operações do CC */
3
4      /*
5         O ficheiro de configuração indicado no método configure() serve para
6         indicar o caminho no
7         sistema atual onde se encontra o módulo PKCS#11 que pretendemos carregar.
8         Por exemplo, num sistema Linux a configuração deve ter o seguinte
9         conteúdo:
10
11         name = PortugaleId
12         library = /usr/local/lib/libpteidpkcs11.so
13
14         Para sistemas Windows a propriedade library deve ter o valor C:\Windows\
15         System32\pteidpkcs11.dll
16         e em MacOS /usr/local/lib/libpteidpkcs11.dylib
17     */
18     Provider p = Security.getProvider("SunPKCS11");
19     p = p.configure(config_file);
20
21     Security.addProvider(p);
22
23     Keystore ks = KeyStore.getInstance ("PKCS11", p);
24     try {
25
26         // Initialize the PKCS#11 token
27         ks.load (null, null);
28
29     }
30     catch (IOException | NoSuchAlgorithmException | CertificateException e) {
31         System.err.println ("Exception while initializing PKCS#11 token:" + e );
32     }
33
34     /* O 2º parâmetro password que para outros módulos seria passado no método
35        getKey()
36        não corresponde ao PIN que protege o uso da chave privada do CC e por
37        isso deve ser passado a null
38        O módulo pteid-pkcs11 vai gerir internamente os pedidos de PIN ao
39        utilizador através de
40        janelas de diálogo idênticas às que são usadas nos métodos de assinatura
41        do SDK, p.ex.
42        PTEID_EIDCard.Sign()
43
44        Se pretender utilizar a chave privada de autenticação a label que se deve
45        passar ao
46        método getKey() será "CITIZEN AUTHENTICATION CERTIFICATE" */
47
48     Key key_handle = ks.getKey("CITIZEN SIGNATURE CERTIFICATE", null);
49
50     /* Os algoritmos de assinatura suportados pelo Cartão de Cidadão via PKCS#11
```

```
        são
42      os seguintes e podem ser especificados através do método Signature.
        getInstance():
43      - SHA512WithRSA
44      - SHA384WithRSA
45      - SHA256withRSA (recomendado por ser compatível com todos os cartões em
        circulação)
46      - SHA512WithRSASSA-PSS
47      - SHA384WithRSASSA-PSS
48      - SHA256withRSASSA-PSS
49      - SHA1withRSA (Não utilizar a não ser por razões de retro-
        compatibilidade com sistemas antigos)
50    */
```

## 8 Compatibilidade com o SDK da versão 1

### 8.1 Métodos removidos

- `ChangeAddress`
- `CancelChangeAddress`
- `GetChangeAddressProgress`
- `GetLastWebErrorCode`
- `GetLastWebErrorMessage`
- `CVC_Authenticate_SM101`
- `CVC_Init_SM101`
- `CVC_WriteFile`
- `CVC_WriteAddr`
- `CVC_WriteSOD`
- `CVC_WriteFile`
- `CVC_R_DH_Auth`
- `CVC_R_Init`

### 8.2 Diferenças no comportamento de alguns métodos

- Método **`pteid.GetCertificates()`**

Na versão anterior 1.26, o certificado «*Baltimore CyberTrust Root*» não está a ser retornado, ao contrário desta versão que obtém tal certificado.

- Método **`pteid.GetPINs()`**

As flags dos PINs retornadas possuem valores diferentes. A versão anterior 1.26, neste momento, retorna o valor 47(base 10) (*0011 0001*)(binário) e esta versão retorna o valor 17(base 10) (*0001 0001*)(binário).

- Método **`pteid.ReadFile()`**

O tamanho do *buffer* retornado com o conteúdo do ficheiro lido tem tamanhos diferentes. A versão 1, retorna em blocos de 240 bytes, enquanto esta versão retorna o tamanho total do ficheiro, que neste momento é de 1000 bytes (para o caso do ficheiro de notas).

- Métodos **`pteid.WriteFile()`** / **`pteid.WriteFile_inOffset()`**

Quando é necessário escrever no ficheiro PersoData (Notas) do Cartão de Cidadão, o pedido de PIN é diferente. Na versão 1, o PIN é pedido uma vez dentro de uma sessão, podendo ser efectuada várias escritas, sem ser pedido novamente o PIN. Nesta versão, o PIN é sempre pedido quando é feita uma nova escrita no ficheiro.

- Métodos **`pteid.VerifyPIN()`**

Na versão 1, quando um PIN é introduzido incorrectamente, é lançada uma excepção de imediato, enquanto que nesta versão tal não acontece. A excepção é apenas lançada se o utilizador escolher a opção "Cancelar" no diálogo de PIN errado que é mostrado.

### 8.3 Códigos de Erro

Em vez de lançar excepções, as funções de compatibilidade para linguagem C mantêm a compatibilidade com versões anteriores em que são devolvidos os códigos descritos nas seguintes tabelas. Os códigos retornados pelo SDK estão apresentados na seguinte tabela, também presentes no ficheiro *eidlibcompat.h*.

Código de retorno	Valor	Descrição
PTEID_OK	0	Função executou com sucesso
PTEID_E_BAD_PARAM	1	Parâmetro inválido
PTEID_E_INTERNAL	2	Ocorreu um erro de consistência interna
PTEID_E_NOT_INITIALIZED	9	A biblioteca não foi inicializada

Foi removida a dependência da biblioteca *pteidlibopensc* contendo um conjunto de códigos de retorno,



descritos na tabela abaixo, continuam a ser mantidos para garantir retrocompatibilidade.

Código de retorno	Valor	Notas
SC_ERROR_NO_READERS_FOUND	-1101	Não existe um leitor conectado
SC_ERROR_CARD_NOT_PRESENT	-1104	O cartão de cidadão não está inserido no leitor
SC_ERROR_KEYPAD_TIMEOUT	-1108	Expirou o tempo para introduzir o PIN
SC_ERROR_KEYPAD_CANCELLED	-1109	O utilizador cancelou a acção de introduzir o PIN
SC_ERROR_AUTH_METHOD_BLOCKED	-1212	O cartão tem o PIN bloqueado
SC_ERROR_PIN_CODE_INCORRECT	-1214	O código PIN ou PUK introduzido está incorrecto
SC_ERROR_INTERNAL	-1400	Ocorreu um erro interno
SC_ERROR_OBJECT_NOT_VALID	-1406	A consistência da informação presente no cartão está comprometida

## 9 Serviços online usados pelo Middleware

Algumas funcionalidades requerem a ligação a serviços online para funcionarem corretamente. É por isso necessário garantir que não existe *firewall* ou outro *software* na rede local que impeça a ligação a estes serviços.

Os *hostnames* e respetivos portos utilizados são listados em seguida por funcionalidade.

### Validação de certificados:

Servidores OCSP:

- ocsf.ecee.gov.pt (porto 80 e 443)
- ocsf.multicert.com (porto 80)
- ocsf.root.cartaodecidadao.pt (porto 80)
- ocsf.auc.cartaodecidadao.pt (porto 80)
- ocsf.asc.cartaodecidadao.pt (porto 80)

Servidores CRL:

- crls.ecee.gov.pt (porto 80)
- pkiroot.multicert.com (porto 80)
- pki.cartaodecidadao.pt (porto 80)

### Selo temporal (por defeito):

- ts.cartaodecidadao.pt (porto 80)

[illegible]