BRySigner SDK 2.1.0.2

Manual

BRY TECNOLOGIA S.A.
FLORIANÓPOLIS - SC

BRy Tecnologia S.A.
Rua: Lauro Linhares, 2123 ï Trindade Center Torre B ï Sala 306.
Bairro: Trindade - CEP: 88.036-002 - Florianópolis ï SC
Fone / Fax: 3234-6696
e-mail: comercial@bry.com.br
Site: **http://www.bry.com.br**

Sumário

1 Introdução 4

1.1 Características Técnicas do Componente 5

1.2 Requisitos mínimos 5

2 Extensão do arquivo assinado 6

3 Componente BRySigner 7

3.1 Nome do componente 7

3.2 Gerando um arquivo .p7s 7

3.3 Comandos OpenSSL para verificação de assinaturas 8

4 Classes 10

4.1 Classe Repositório (ClassID: 5B12C0E6-F4E6-4a74-9314-89AFA21AA1C5) 10

4.2 Classe Certificado (ClassID: 86DE8D4F-D663-4395-82E9-051D79766FB9) 15

4.3 Classe Assinador (ClassID: D74AA60A-D77B-4482-8850-4768E636AAC7) 57

4.4 Classe Verificador (ClassID: 1690EE68-8D31-473d-9CEB-C51D56A76D7B) 67

4.5 Classe Carimbo (ClassID: 83C5AF6B-0CEF-4A58-9ACC-009ABEF94260) 84

4.6 Classe CRL (ClassID: 86DE8D4F-D663-4395-82E9-051D79766FB9) 91

Apêndice A 101

Apêndice B 102

1 Introdução

O kit de desenvolvimento de sistemas BRySigner SDK 2.1.0.2 (Software Development Kit) provê uma interface que propicia a construção de softwares que interajam com a assinatura digital de documentos eletrônicos. Ele é formado por um componente que implementa classes especializadas para gerar o resumo criptográfico da informação, assinar digitalmente e verificar os documentos já assinados.

De uma forma geral, uma aplicação desenvolvida com o BRySigner SDK segue a organização e interação com os objetos criptográficos conforme apresentado na figura 1.

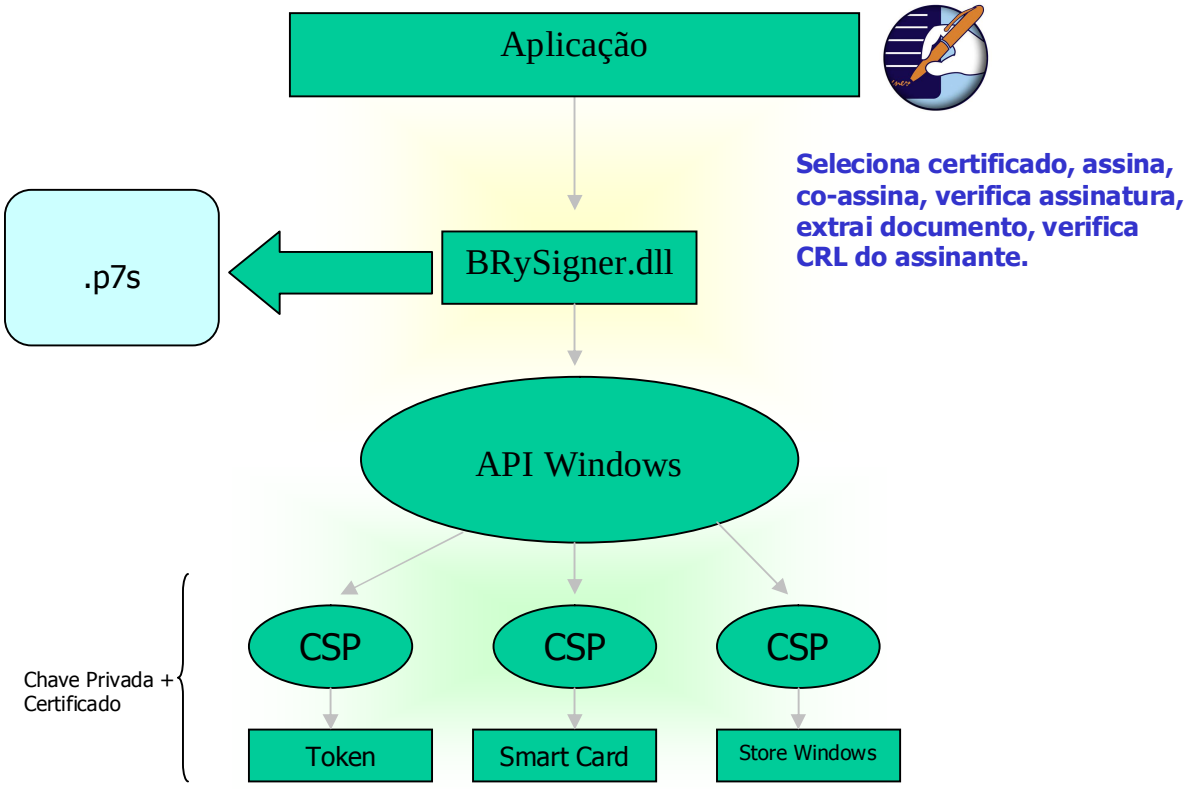

Figura 1 – Estrutura do BRySigner SDK.

O componente BRySigner é construído em camadas. Os provedores de serviço de criptografia (CSP) do Windows gerenciam a instalação e uso dos certificados digitais e o BRySigner, através da API do Windows acessa, de forma genérica os CSPs, que por sua vez, habilitam a assinatura digital dos documentos eletrônicos. Os certificados devem obedecer ao padrão x509v3.

O BRySigner SDK conta com as seguintes funcionalidades:

- Assinar
  - **Assinatura única:** Assinatura digital única que identifica o remetente de uma mensagem digital.
  - **Assinatura múltipla:** Só feita em documentos que já foram assinados pelo menos uma vez, garantindo que o documento é confiado por mais de uma entidade.
- Verifica assinatura
- Extrai documento assinado
- Extrair informações de certificados digitais
- Manipulação de certificados contidos nos repositórios gerenciados pelo Windows.
- Assina arquivo no formato Detached
- Verifica assinatura no formato Detached
- Assina no formato Detached registrado em memória
- Verifica assinatura no formato Detached registrado em memória
- Extrai **carimbo de tempo** da assinatura

1.1 Características Técnicas do Componente

O BrySigner SDK é composto por uma biblioteca de acesso dinâmico (.dll) desenvolvida para utilização em ambiente Windows.

Desenvolvido em Visual Studio C++, o BRySigner SDK é um componente COM, sendo ideal para aplicativos internos e para uso na Web.

Gera assinaturas em padrão PKCS#7;

Trabalha com certificado(s) digital(is) padrão x509v3 instalado em Windows, token ou em cartões inteligentes (smart card);

Biblioteca disponível para integração com as mais diversas linguagens entre elas Delphi, C/C++, Java Script, .Net, ASP, Visual Basic, Lótus Notes, etc.

O registro do componente o Windows não irá sobrescrever a versão anterior, possibilitando a utilização de ambas. Para utilizar esta nova versão atualize o class-id.

1.2 Requisitos mínimos

Plataforma Windows 2000 ou superior atualizados com o último pacote de serviços (Service Pack) disponível;

Internet Explorer 5.5 ou superior.

2 Extensão do arquivo assinado

A definição das extensões adotadas pela BRy basearam-se na RFC 2311 (S/MIME Version 2 Message Specification). O quadro 1 ilustra as extensões utilizadas pelo S/MIME, descrita nesta RFC. O formato .p7m serve tanto para signedData quanto para envelopedData, sendo que a definição do seu tipo (smime-type=enveloped-data ou smime-type=signed-data) é descrita no cabeçalho do arquivo.

MIME Type	File Extension
application/pkcs7-mime (envelopedData)	.p7m
application/pkcs7-mime (degenerate signedData "certs-only" message)	.p7c
application/pkcs7-signature	.p7s
application/pkcs10	.p10

Quadro 1. http://www.ietf.org/rfc/rfc2311.txt (S/MIME Version 2 Message Specification)

O quadro 2 ilustra as extensões adotadas nos aplicativos da BRy. Ele difere-se do quadro 1 nos seguintes aspectos:

- O .p7s é um documento assinado que contém o original e a assinatura juntas, o equivalente ao .p7m.
- O .p7m contém um arquivo cifrado ou cifrado e assinado.
- O .sig contém apenas a assinatura, deixando o conteúdo em texto plano, equivalente ao .p7s, só que sem o documento.
- O .brysig é o documento original junto com o .sig, armazenado distintamente em um arquivo compactado. Para ler o documento original basta descompactar o arquivo e abrir o original.

Tipo	Extensão de arquivo
application/pkcs7-envelopedData	.p7m
application/pkcs7-signedData	.p7s
application/pkcs7-signature (somente assinatura)	.sig
application/pkcs7-bry (contém o documento original e a assinatura (.sig) em um arquivo compactado)	.brysig

Quadro 2. Extensão de arquivo adotada nos aplicativos da BRy.

3 Componente BRySigner

3.1 Nome do componente

BrySignerCOM.DLL (Windows)

3.2 Gerando um arquivo .p7s

O arquivo .p7s tem o tamanho do arquivo original acrescido da assinatura, dos certificados utilizados e atributos da assinatura.

Em cada assinatura é adicionado o atributo "Document type" no campo "authenticatedAttributes" da estrutura "SignerInfo", refletindo o nome e extensão do arquivo assinado (OID 1.3.6.1.4.1.311.88.2.1), também é possível adicionar o atributo "Document description" no campo "authenticatedAttributes" da estrutura "SignerInfo", refletindo uma descrição sobre o documento assinado (OID 1.3.6.1.4.1.311.88.2.2).

Cada assinatura é executada considerando o atributo "signingTime" no campo "authenticatedAttributes" da estrutura "SignerInfo", refletindo a data-hora de sua efetivação, sendo que todas as co-assinaturas estarão no mesmo nível na estrutura do PKCS#7, pela repetição do campo "SignerInfo" da estrutura "SignedData", tantas vezes quantas forem o número de co-assinaturas digitais apostas. A data-hora de efetivação é obtida do sistema operacional da estação cliente ou informada através de uma função a biblioteca BRySigner.

De um arquivo .p7s não é possível extrair a assinatura e guardá-la em um arquivo separado, porque o conteúdo do arquivo original faz parte da estrutura PKCS#7. Neste formato pode-se apenas extrair o documento original e salvá-lo em um arquivo.

A assinatura gerada pelo componente BRySigner segue o padrão PKCS#7 codificada no formato DER, ou seja, binário. Nas assinaturas em blocos de memória, a assinatura estará codificada em Hexadecimal ou Base 64, dependendo da opção selecionada pelo desenvolvedor.

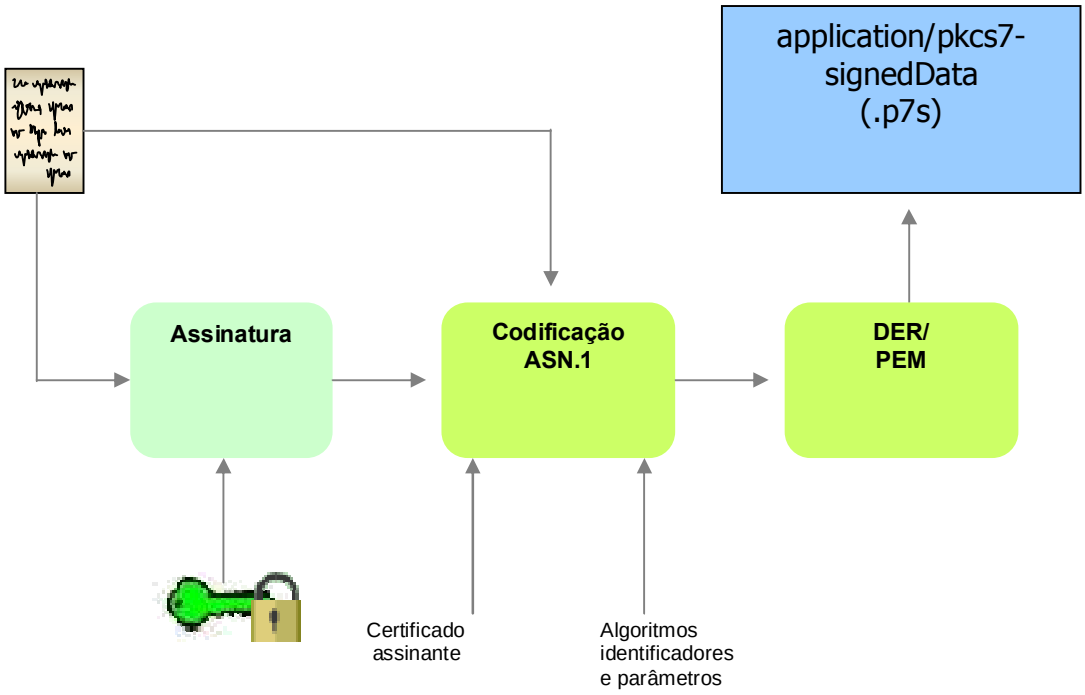

Figura 2 – Gerando um arquivo .p7s

3.3 Comandos OpenSSL para verificação de assinaturas

O padrão de assinatura utilizado pela BRy, o PKCS#7 pode ser verificado diretamente com as funções da OpenSSL.

Abaixo estão descritas as funções da OpenSSL que podem ser utilizadas para esta finalidade:

Verifica assinatura e extrai o conteúdo assinado:

```
openssl smime -inform DER -in arquivo_assinado.txt.p7s -verify –noverify
–out arquivo_extraido
```

(a cláusula ï noverify é colocada para não proceder à verificação do assinante do certificado digital. Se quiser colocar ela, o certificado raiz deve ser especificado).

Comando para extrair os certificados utilizados para assinar:

```
openssl pkcs7 -in arquivo_assinado.txt.p7s -inform DER -print_certs -out
certs.pem
```

Comando para verificar:

```
openssl smime -in arquivo_assinado.txt.p7s -verify -noverify -content
conteudooriginal -inform DER
```

Comando para passar um arquivo assinado no formato DER para PEM:

```
openssl pkcs7 -inform DER -outform PEM -in ArquivoAss -out saida
```

Comando para calcular o hash:

```
openssl dgst -SHA1 -binary -out ArqHash Arquivo
```

Comando para retirar um certificado de um arquivo assinado:

```
openssl pkcs7 -inform PEM -in Assinado.pem -out certificado -print_certs
```

Comando para extrair chave publica de um certificado:

```
openssl x509 -in <cert> -pubkey -noout >chavepublica.pem
```

Comando para passar o arquivo assinado em asn1:

```
openssl asn1parse -inform PEM -in Assinado.pem |more
```

Comando para extrair o hash assinado do arquivo assinado:

```
openssl asn1parse -in Assinado.pem -out hashAssinado -noout -strparse
NodoHash
```

Comando para verificar o hash assinado:

```
openssl rsautl -in hashAssinado -out dadoverificado -inkey chavepub.pem -
pubin -verify
```

Cabe salientar que um arquivo DER é binário, enquanto que o PEM esta em base64. Isto significa que o tamanho do arquivo PEM é maior, podendo ser visualizado no exemplo abaixo:

```
-rw-r--r--    1 suporte  suporte      1443 Jan  8 13:09 teste.der
-rw-rw-r--    1 suporte  suporte      1997 Jan  9 11:52 teste.pem
```

Portanto, é importante utilizar uma função para codificar DER→PEM e PEM→DER.

Referência: http://www.openssl.org/docs/apps/smime.html#

4 Classes

O componente BrySigner possui as seguintes classes:

- Repositorio
- Certificado
- Assinador
- Verificador
- Carimbo
- CRL

4.1 Classe Repositório (ClassID: 5B12C0E6-F4E6-4a74-9314-89AFA21AA1C5)

A classe Repositório foi desenvolvida com a tecnologia COM. Esta classe tem a finalidade de representar um determinado ñStoreò do Windows, este ñStoreò pode ser o ñMYò, ñAddressBookò, ñCAò, ñRootò, ou qualquer outro que esteja criado no Windows.

Os certificados pertencentes ao Repositório inicializado poderão ser visualizados e ter seus dados extraídos.

Também é possível instalar certificados no Repositório aberto através do certificado em hexadecimal, assim como importar certificados que estão no formato de arquivo de certificados da BRy. Este formato será apresentado logo a seguir da função de instalação dos certificados.

Os métodos implementados nesta classe estão descritos a seguir:

MÉTODO: inicialize

Este é o inicializador da classe Repositório. É preciso passar o local do repositório que se deseja abrir/inicializar (MY, AddresBook, CA, Root), e o armazenamento do repositório.

O armazenamento, pode assumir os seguintes valores:

0 : utiliza CERT_SYSTEM_STORE_CURRENT_USER (certificados do usuário)

1 : utiliza CERT_SYSTEM_STORE_LOCAL_MACHINE (certificados da máquina)

Assinatura do método:

inicialize(BSTR local, int armazenamento);

Parâmetros:

Local	BSTR	Local do Store a ser aberto
Armazenamento	int	É o local onde os certificados estão armazenados, normalmente se utiliza CERT_SYSTEM_STORE_CURRENT_USER

Retorno: Este método não tem retorno.

MÉTODO: finalize

Este é o destrutor da classe Repositório. Quando este método é executado o objeto não pode mais ser utilizado, ao menos que outro objeto seja criado. Ele limpa todos os dados que foram carregados. Sempre ao final da utilização de um objeto da classe Repositório, esta função deve ser chamada para desalocar a memória alocada, e destruir os objetos internos criados pelo objeto Repositório.

Assinatura do método:

finalize();

Parâmetros: Este método não tem parâmetros.

Retorno: Este método não tem retorno.

MÉTODO: adicionar

Adiciona um novo certificado ao repositorio com o local e o nome que foram especificados na inicialização do objeto da classe repositório.

Assinatura do método:

int adicionar(BSTR certificado, int FLAG);

Parâmetros:

certificado	BSTR	Valor do certificado. O valor esta em um dos formatos indicados pela flag
Flag	int	Flag:

		0 - então certificado indica um nome de arquivo (no formato DER) 1 - então o parâmetro certificado é o conteúdo do certificado em hexadecimal 2 - então o parâmetro certificado é o conteúdo do certificado em hexadecimal

Retorno:

Retorno	int	**sucesso:** 1 **Erros comuns:** ERRO_PONTEIRO_STORE_NULO ERRO_HEX_BIN ERRO_ABRIR_ARQUIVO ERRO_ADICIONAR_CERT_STORE ERRO_BASE64_BIN ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: adicionarDoArquivo

Adiciona os certificados contidos no arquivo, desde que este esteja no formato que a BRy define no apêndice A, ao store identificado dentro do arquivo, independente do store aberto no momento da inicialização do objeto desta classe.

Assinatura do método:

int adicionarDoArquivo(BSTR arquivo);

Parâmetros:

Arquivo	BSTR	caminho completo do arquivo que contém os certificados no formato BRy

Retorno:

Retorno	int	**sucesso:** 1 **Erros comuns:** ERRO_ABRIR_ARQUIVO ERRO_FORMATO_ARQUIVO_BRY ERRO_HEX_BIN ERRO_ADICIONAR_CERT_STORE ERRO_PARAMETRO_TIPO_INVALIDO ... *verificar valores de retorno dos erros na tabela de erros

***Obs: O formato do arquivo de certificados BRy pode ser encontrado no Apêndice A.

MÉTODO: getCountCertificados

Obtém o número de certificados contidos no repositorio aberto pelo método inicialize.

Assinatura do método:

int getCountCertificados ():

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de certificados no repositório

MÉTODO: getCertificado

Retorna um objeto COM do Certificado na posição passada como parâmetro, na lista de certificados do repositório. Com o objeto retornado, pode-se recuperar os dados do certificado, assim como usá-lo para assinar/co-assinar um documento.

Assinatura do método:

ICertificado getCertificado(int posicao):

Parâmetros:

posição	int	posição do certificado na lista de certificados do repositório aberto iniciando da posição 0.

Retorno:

Retorno	ICertificado	**Sucesso:** objeto COM da classe Certificado **Erro:** NULL

MÉTODO: getCertificadoBase64

Retorna um certificado no formato Base64 na posição passada como parâmetro, na lista de certificados do repositório.

Assinatura do método:

string getCertificado(int posicao):

Parâmetros:

posição	int	posição do certificado na lista de certificados do repositório aberto iniciando da posição 0.

Retorno:

Retorno	string	**Sucesso:** certificado base64

MÉTODO: existeCartaoLeitora

Retorna um booleano contendo informação se existe uma leitora de cartão sendo utilizada no computador e se a mesma possui um cartão em uso.

Assinatura do método:

boolean existeCartaoLeitora ():

Retorno:

Retorno	boolean	**Sucesso:** Existe um cartão em uso no momento

4.2 Classe Certificado (ClassID: 86DE8D4F-D663-4395-82E9-051D79766FB9)

A classe Certificado foi desenvolvida com a tecnologia COM. Ela tem a finalidade de representar um determinado Certificado do Windows, podendo então a partir de um determinado objeto desta classe trazer as informações do certificado, como por exemplo, o assunto do certificado, o emissor, as finalidades, etc..

Um objeto da classe Certificado também é utilizado pela classe Verificador e pode ser necessário para extrair o "hash", ou resumo criptográfico, do certificado e passar para a classe Assinador que localizará o certificado no Store do Windows através deste "hash"

Na classe Assinador, é necessário ter um "hash", para que um determinado documento que venha a ser assinado ou co-assinado, utilizando o objeto certificado pode-se então extrair este "hash" para que a assinatura possa ser realizada. No entanto, se o usuário conhecer o "hash" do certificado que será utilizado na assinatura, não será necessário instanciar um objeto da classe Certificado.

Já na classe Verificador, após ser efetuada a verificação da assinatura, um objeto certificado é retornado, e então os dados do certificado que assinou o documento podem ser extraídos/exibidos.

MÉTODO: inicialize

Este é o construtor da classe Certificado. É preciso passar o "hash" do certificado a ser inicializado, pois é através dele que os dados do certificado serão recuperados. Este método não deverá ser chamado nos casos onde um objeto da classe Certificado retorna na chamada de uma função, como por exemplo, o método getCertificado da classe Repositório.

obs - é necessário que o certificado esteja instalado na máquina local.

Assinatura do método:

inicialize(BSTR hash);

Parâmetros:

Hash	BSTR	hash SHA-1 em hexadecimal do certificado ao qual se deseja inicializar um objeto. Não deve possuir espaços entre os valores do hash.

Retorno: Este método não tem retorno.

MÉTODO: inicializeCertBase64

Este é o construtor da classe Certificado. É preciso passar o Certificado no formato Base64 a ser inicializado, pois é através dele que os dados do certificado serão recuperados. Este método não deverá ser chamado nos casos onde um objeto da classe Certificado retorna na chamada de uma função, como por exemplo, o método getCertificado da classe Repositório.

obs - é necessário que o certificado esteja instalado na máquina local.

Assinatura do método:

inicialize(BSTR __strCertBase64);

Parâmetros:

strCertBase64	BSTR	Certificado no formato Base64

Retorno: Este método não tem retorno.

MÉTODO: finalize

Este é o destrutor da classe Certificado. Quando este método é executado o objeto não pode mais ser usado, ao menos que outro objeto seja criado. Ele limpa todos os dados que foram carregados. Sempre ao final da utilização de um objeto da classe Certificado que chamou a função inicialize, esta função deve ser chamada para desalocar a memória alocada, e destruir os objetos internos criados pelo objeto Certificado. Nos casos onde o objeto da classe Certificado é retornado de um método de outras classes, como por exemplo, o método getCertificado da classe Repositório, não deve-se chamar esta função, pois a própria função de finalização da classe que retornou o objeto Certificado, realizará a finalização.

Assinatura do método:

finalize();

Parâmetros: Este método não tem parâmetros.

Retorno: Este método não tem retorno.

MÉTODO: mostrarCertificado

Mostra a tela do certificado que é representado por este objeto. Este método chama a janela do Windows padrão de exibição do certificado.

Assinatura do método:

int mostrarCertificado ();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Sucesso:** 1 **Erros comuns:** ERRO_PONTEIRO_CERTIFICADO_NULO ERRO_MOSTRAR_CERTIFICADO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: getIdCertificado

Retorna o valor do hash do certificado que o objeto da classe representa.

Assinatura do método:

BSTR getIdCertificado();

Parâmetros: Esta função não tem parâmetros.

Retorno:

retorno	BSTR	**Sucesso:** Retorna o hash sha-1 em hexadecimal do certificado **Erro:** retorna vazio

MÉTODO: getAssunto

Retorna o valor do Assunto (CN,O,OU,L,S,C,E) do certificado que o objeto representa.

Assinatura do método:

BSTR getAssunto();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoCN

Retorna o valor do CN (Nome Comum) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoCN();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CN do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoO

Retorna o valor do O (Organização) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoO();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o O do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoOU

Retorna o valor do OU (Unidade da Organização) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoOU();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o OU do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoL

Retorna o valor do L (Localidade) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o L do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoS

Retorna o valor do S (Estado) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoS();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o S do Assunto do certificado

		Falha: retorna vazio

MÉTODO: getAssuntoC

Retorna o valor do C (País) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoC();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o C do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getAssuntoE

Retorna o valor do E (e-mail) do Assunto do certificado que o objeto representa

Assinatura do método:

BSTR getAssuntoE();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o E do Assunto do certificado **Falha:** retorna vazio

MÉTODO: getDataNascimentoTitular

Retorna o valor da data de nascimento do portador ou responsável do certificado desde que este certificado seja emitido por autoridades certificadoras ICP-Brasil, que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.1 ou OID = 2.16.76.1.3.4.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getDataNascimentoTitular();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna a data de nascimento contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getCPF

Retorna o valor do CPF do portador ou responsável do certificado desde que este certificado seja emitido por autoridades certificadoras ICP-Brasil, que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.1.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getCPF();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CPF contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getNIS

Obtém o NIS (PIS, PASEP ou CI) da pessoa física ou jurídica para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto

(ñSubject Alternative Name) com o campo outro nome (ñotherName) de OID = 2.16.76.1.3.1 ou OID = 2.16.76.1.3.4.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getNIS();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o NIS contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getRG

Retorna o valor do RG do portador ou responsável do certificado desde que este certificado seja emitido por autoridades certificadoras ICP-Brasil, que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Name) com o campo outro nome (ñotherName) de OID = 2.16.76.1.3.1 ou OID = 2.16.76.1.3.4.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getRG();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o RG contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getOrgaoExpedidor

Obtém o órgão expedidor do RG do portador ou responsável do certificado desde que este certificado seja emitido por autoridades certificadoras ICP-Brasil, que emitem certificados contendo a extensão nome alternativo do assunto

(ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.1 ou OID = 2.16.76.1.3.4.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getOrgaoExpedidor();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o órgão expedidor seguido de sua UF contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getCEIPF

Obtém o número do Cadastro Específico do INSS (CEI) da pessoa física titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.6.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Se o certificado é de pessoa jurídica, utilize o método getCEIPJ.

Assinatura do método:

BSTR getCEIPF();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CEI contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getCEIPJ

Obtém o número do Cadastro Específico do INSS (CEI) da pessoa jurídica titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.7.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Se o certificado é de pessoa física, utilize o método getCEIPF.

Assinatura do método:

BSTR getCEIPJ();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CEI contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getTituloEleitor

Obtém o número do Título de Eleitor da pessoa física titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.5.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Os métodos getZonaEleitoral, getSecaoEleitoral e getMunicipioUF podem ser utilizados para obter as demais informações do Título de Eleitor do titular do certificado.

Assinatura do método:

BSTR getTituloEleitor();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o número do Título de

		Eleitor contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getZonaEleitoral

Obtém a Zona Eleitoral do título de eleitor da pessoa física titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.5.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Os métodos getTituloEleitor , getSecaoEleitoral e getMunicipioUF podem ser utilizados para obter as demais informações do Título de Eleitor do titular do certificado.

Assinatura do método:

BSTR getZonaEleitoral();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna a Zona Eleitoral contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getSecaoEleitoral

Obtém a Seção Eleitoral do Título de Eleitor da pessoa física titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.5.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Os métodos getZonaEleitoral, getTituloEleitor e getMunicipioUF podem ser utilizados para obter as demais informações do Título de Eleitor do titular do certificado.

Assinatura do método:

BSTR getSecaoEleitoral();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna a Seção Eleitoral contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getMunicipioUF

Obtém o Município seguido da UF do Título de Eleitor da pessoa física titular do certificado, para certificados ICP-Brasil, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.5.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Os métodos getZonaEleitoral, getTituloEleitor e getSecaoEleitoral podem ser utilizados para obter as demais informações do Título de Eleitor do titular do certificado. Assinatura do método:

BSTR getMunicipioUF();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o Município seguido da UF contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getNomeResponsavel

Obtém o Nome do Responsável pelo certificado, para certificados ICP-Brasil de pessoa jurídica, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.2.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getNomeResponsavel();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o Nome do Responsável seguido da UF contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getCNPJ

Obtém o CNPJ da empresa a qual o certificado foi emitido, para certificados ICP-Brasil de pessoa jurídica, desde que este seja emitido por autoridades certificadoras que emitem certificados contendo a extensão nome alternativo do assunto (ñSubject Alternative Nameò) com o campo outro nome (ñotherNameò) de OID = 2.16.76.1.3.3.

Para extrair esta informação foi seguido o padrão determinado pelo comitê gestor da ICP-Brasil através das resoluções nº31 de 29 de janeiro de 2004 e nº35 de 21 de outubro de 2004.

Assinatura do método:

BSTR getCNPJ ();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CNPJ da empresa contido na extensão nome alternativo do assunto **Falha:** retorna vazio

MÉTODO: getCountPoliticas

Obtém o número de políticas do certificado contidas na extensão Diretivas dos Certificados (ñCertificate Policiesò).

Assinatura do método:

int getCountPoliticas():

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de política do certificados

MÉTODO: getPoliticaOID

Este método tem o objetivo de recuperar o OID da política na posição passada como parâmetro, na lista de políticas do certificado contidas na extensão Diretivas dos Certificados (ñCertificate Policiesò).

Assinatura do método:

BSTR getPoliticaOID(int posicao);

Parâmetros:

posicao	int	posição da lista de políticas iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** valor do OID da política se existir **Erro:** vazio

MÉTODO: getPoliticaCPS

Este método tem como objetivo obter o valor da política CPS (Declaração de Práticas de Certificação), ou seja, obtém a URI do documento que contém a DPC (ou CPS) da política em questão.

Assinatura do método:

BSTR getPoliticaCPS(int posicao);

Parâmetros:

posicao	Int	posição da lista de políticas iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** a URI do documento que especifica a Declaração de Práticas de Certificação da política em questão

Erro: vazio

MÉTODO: getCountOIDSubjectAlternativeName

Obtém o número de OIDs contidos estrutura Subject Alternative Name do certificado.

Assinatura do método:

int getCountOIDSubjectAlternativeName();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de OIDs do SAN do certificados

MÉTODO: getOIDSubjectAlternativeName

Este método tem como objetivo obter o valor do OID do Subject Alternative Name do certificado.

Assinatura do método:

BSTR getOIDSubjectAlternativeName (int posicao);

Parâmetros:

posicao	int	posição da lista de OID iniciando da posição 0.

Retorno:

Retorno	BSTR	**Sucesso:** o valor em string do OID do certificado. **Erro:** vazio

MÉTODO: getEmissor

Retorna o valor do Emissor (CN,O,OU,L,S,C,E) do certificado que o objeto representa

Assinatura do método:

BSTR getEmissor();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorCN

Retorna o valor do CN (Nome Comum) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorCN();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CN do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorO

Retorna o valor do O (Organização) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorO();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o O do Emissor do

		certificado **Falha:** retorna vazio

MÉTODO: getEmissorOU

Retorna o valor do OU (Unidade da Organização) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorOU();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o OU do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorL

Retorna o valor do L (Localidade) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o L do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorS

Retorna o valor do S (Estado) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorS();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o S do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorC

Retorna o valor do C (País) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorC();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o C do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getEmissorE

Retorna o valor do E (e-mail) do Emissor do certificado que o objeto representa

Assinatura do método:

BSTR getEmissorE();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o E do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getRFC822Name

Retorna o valor do E (e-mail) do Emissor do certificado codificado no atributo RFC822name.

Assinatura do método:

BSTR getRFC822Name ();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o e-mail do Emissor do certificado **Falha:** retorna vazio

MÉTODO: getNumeroSerial

Retorna o valor do Número Serial do certificado que o objeto representa

Assinatura do método:

BSTR getNumeroSerial();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Número Serial do certificado **Falha:** retorna vazio

MÉTODO: getDataInicio

Retorna o valor da data de emissão do certificado que o objeto representa

Assinatura do método:

BSTR getDataInicio():

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** data de emissão do certificado

		Falha: retorna vazio

MÉTODO: getDataTermino

Retorna o valor da data de vencimento do certificado que o objeto representa

Assinatura do método:

BSTR getDataTermino();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** data de vencimento do certificado **Falha:** retorna vazio

MÉTODO: existeExtensao

Nesta função o retorno é verdadeiro se a extensão cujo OID passado como parâmetro esta presente no certificado. Caso o OID não esteja presente o retorno é 0.

Assinatura do método:

BOOL existeExtensao (BSTR OID);

Parâmetros:

OID	BSTR	identificador da extensão a ser pesquisada

Retorno:

retorno	BOOL	**Existe:** 1 **Não existe:** 0

MÉTODO: existeFinalidade

Nesta função o retorno é verdadeiro se a finalidade cujo OID passado como parâmetro esta presente no certificado. Caso o OID não esteja presente o retorno é 0.

Assinatura do método:

BOOL existeFinalidade (BSTR OID);

Parâmetros:

OID	BSTR	identificador da finalidade a ser pesquisada

Retorno:

retorno	BOOL	**Existe:** 1 **Não existe:** 0

MÉTODO: existeUsoChave

Nesta função o retorno é verdadeiro se o Uso da Chave cujo OID passado como parâmetro esta presente no certificado. Caso o OID não esteja presente o retorno é 0.

Assinatura do método:

BOOL existeUsoChave(BSTR OID);

Parâmetros:

OID	BSTR	identificador do Uso da Chave a ser pesquisada

Retorno:

retorno	BOOL	**Existe:** 1 **Não existe:** 0

MÉTODO: existeCRL

O objetivo desta função é verificar se a CRL do certificado, esta instalada na máquina corrente e é válida. Caso a CRL não esteja instalada na máquina ou a CRL instalada é inválida, o retorno será um valor de erro.

Assinatura do método:

Int existeCRL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Existe:** 1 **Erros comuns:** ERRO_PONTEIRO_CERTIFICADO_NULO ERRO_CARREGANDO_CERTIFICADO_EMISSOR ERRO_CRL_AINDA_NAO_VALIDA ERRO_CRL_EXPIRADA ERRO_CRL_NAO_INSTALADA ERRO_CARREGANDO_CRL ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: getCountCRL

Recupera o número de pontos de distribuição da CRL do certificado em questão.

Assinatura do método:

Int getCountCRL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de pontos de distribuição da CRL

MÉTODO: verificarCRL

Verifica o status do certificado quanto a CRL, indicando se o mesmo é válido ou não. Este método realiza a busca na CRL para verificar se o certificado está revogado.

Assinatura do método:

Int verificarCRL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Certificado OK:** 1 **Erros comuns:** ERRO_PONTEIRO_CERTIFICADO_NULO ERRO_CARREGANDO_CERTIFICADO_EMISSOR ERRO_CRL_AINDA_NAO_VALIDA ERRO_CRL_EXPIRADA ERRO_CRL_NAO_INSTALADA

ERRO_CARREGANDO_CRL
ERRO_CERTIFICADO_REVOGADO
...
*verificar valores de retorno dos erros na tabela de erros

MÉTODO: baixarCRL

Esta função verifica onde se encontra a CRL do certificado (endereço de distribuição), verifica se o computador está conectado, e se estiver, baixa a CRL e a instala na máquina. Esta função só funciona para o protocolo http, sendo que apenas os endereços http é que serão baixados. Se o computador em questão implementa proxy em seu acesso a internet, o SDK permite setar o login e senha através dos métodos setLoginProxy e setSenhaProxy, caso estes valores não sejam informados, uma janela do pedido de autenticação pelo proxy será exibida de modo que o usuário deverá digitar seu login e senha. Como pode ocorrer de uma janela ser exibida para solicitação de login e senha, esta função deve ser utilizada com cuidado em servidores, sendo que se estes utilizam proxy, o login e a senha devem obrigatoriamente estar setados através das funções setLoginProxy e setSenhaProxy.

Assinatura do método:

Int baixarCRL():

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**CRL Baixada e Instalada:** 1 **Erros Comuns:** ERRO_CARREGANDO_CRL ERRO_CAMINHO_CRL ERRO_IDENTIFICACAO_URL ERRO_LENDO_ARQUIVO_CRL ERRO_ABRINDO_ARQUIVO_CRL ERRO_INSTALANDO_CRL ERRO_WSASTARTUP_FAILED ERRO_WSANOTINITIALISED ERRO_WSAENETDOWN ERRO_WSAHOST_NOT_FOUND ERRO_WSATRY_AGAIN ERRO_WSANO_RECOVERY ERRO_WSANO_DATA ERRO_WSAEINPROGRESS ERRO_WSAEAFNOSUPPORT ERRO_WSAEFAULT ERRO_WSAEINTR ERRO_ACESSO_NEGADO ERRO_AUTENTICACAO_NO_PROXY

ERRO_NO_SERVIDOR
ERRO_ACESSO_PROIBIDO
ERRO_PAGINA_NAO_ENCONTRADA
ERRO_CONECTANDO_URL
ERRO_TIMEOUT
ERRO_URL_INVALIDA
ERRO_CONSULTA_INVALIDA
ERRO_RESPOSTA_SERVIDOR
ERRO_SERVIDOR_REQUER_CERTIFICADO_CLIENTE
ERRO_CONEXAO_TERMINADA
ERRO_CONEXAO_RESETADA
ERRO_INTERNO_INTERNET
...

*verificar valores de retorno dos erros na tabela de erros

MÉTODO: setLoginProxy

Seta qual deverá ser o login do usuário que deverá ser utilizado pela função baixarCRL no caso de utilização de proxy.

Assinatura do método:

int setLoginProxy(BSTR __login);

Parâmetros:

__login	BSTR	é o login que deverá ser utilizado pelo proxy na função baixarCRL

Retorno:

retorno	int	**Sucesso:** 1 **Erro:** esta função não retorna erro

MÉTODO: setSenhaProxy

Seta qual deverá ser a senha do usuário que deverá ser utilizado pela função baixarCRL no caso de utilização de proxy.

Assinatura do método:

int setSenhaProxy(BSTR __senha);

Parâmetros:

__senha	BSTR	é a senha que deverá ser utilizada pelo proxy na função baixarCRL

Retorno:

retorno	int	**Sucesso:** 1 **Erro:** esta função não retorna erro

MÉTODO: verificarValidade

Verifica a validade do certificado, indicando se o tempo do mesmo é válido ou não. Com esta função é possível verificar se o certificado está expirado ou ainda não é válido.

Assinatura do método:

int verificarValidade();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Certificado válido:** 1 **Erros comuns:** ERRO_PONTEIRO_CERTIFICADO_NULO ERRO_CERTIFICADO_AINDA_NAO_VALIDO ERRO_CERTIFICADO_EXPIRADO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: status

Verifica o status do certificado, indicando se o mesmo é válido ou não. Esta função é equivalente à chamada dos métodos verificarValidade e verificarCRL, sendo que o mesmo efetua as duas verificações e indica se o certificado é válido quanto sua validade e se não está presente na CRL.

Assinatura do método:

int status();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Certificado válido:** 1 **Erros comuns:** ERRO_PONTEIRO_CERTIFICADO_NULO ERRO_CERTIFICADO_AINDA_NAO_VALIDO ERRO_CERTIFICADO_EXPIRADO ERRO_CARREGANDO_CERTIFICADO_EMISSOR ERRO_CRL_AINDA_NAO_VALIDA ERRO_CRL_EXPIRADA ERRO_CRL_NAO_INSTALADA ERRO_CARREGANDO_CRL ERRO_CERTIFICADO_REVOGADO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: getCountUsoChaves

Recupera o número de usos da chave possíveis para o certificado em questão.

Assinatura do método:

int getCountUsoChaves();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de usos da chave do certificado

MÉTODO: getCountExtensões

Recupera o número de extensões presentes no certificado.

Assinatura do método:

int getCountExtensoes();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de extensões presentes no certificado

MÉTODO: getCountFinalidades

Recupera o número de finalidades do certificado em questão.

Assinatura do método:

int getCountFinalidades();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	número de finalidades do certificado

MÉTODO: getEndCRL

Recuperar o endereço de distribuição da CRL. Um certificado pode ter mais do que um ponto de distribuição da CRL, portanto, com o parâmetro posição, você obtém o endereço da CRL na posição passada.

Assinatura do método:

BSTR getEndCRL(int posicao);

Parâmetros:

posicao	int	posição da lista de pontos de distribuição da CRL iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** valor do endereço de distribuição da CRL se existir **Erro:** vazio

MÉTODO: getUsoChaves

Recupera o uso da chave na posição, passada como parâmetro, na lista de usos da chave do certificado. Os valores dos OIDs possíveis podem ser encontrados logo abaixo da definição da função.

Assinatura do método:

BSTR getUsoChaves(int posicao);

Parâmetros:

posicao	int	posição da lista de uso chaves iniciando da posição 0

Retorno:

retorno	BSTR	**Sucesso:**valor do OID do uso da chave se existir **Erro:** vazio

Valores que o OID podem assumir, e seus respectivos significados:

OID	SIGNIFICADO
128	Assinatura Digital
64	Não Repúdio
32	Codificação de Chaves
16	Codificação de dados
8	CERT_KEY_AGREEMENT_KEY_USAGE
4	CERT_KEY_CERT_SIGN_KEY_USAGE
2	CERT_OFFLINE_CRL_SIGN_KEY_USAGE

MÉTODO: getFinalidades

Recupera a finalidade na posição, passada como parâmetro, na lista de finalidades do certificado. A finalidade representa a extensão de uso avançado da chave no certificado digital.

Assinatura do método:

BSTR getFinalidades(int posicao);

Parâmetros:

posicao	int	posição da lista de finalidades iniciando da posição 0

Retorno:

retorno	BSTR	**Sucesso:**valor do OID da finalidade na posição se existir **Erro:** vazio

MÉTODO: getExtensaoValor

Recupera o valor da extensão na posição passada como parâmetro, na lista de extensões do certificado. O valor da extensão é o valor codificado, por isso é necessário conhecer a estrutura da extensão para se poder retornar os valores desejados corretamente.

Assinatura do método:

BSTR getExtensaoValor(int posicao);

Parâmetros:

posicao	int	posição da lista de Extensões iniciando da posição 0

Retorno:

retorno	BSTR	**Sucesso:**valor da extensão na posição se existir **Erro:**vazio

MÉTODO: getExtensaoOID

Recupera o OID da extensão na posição passada como parâmetro, na lista de extensões do certificado.

Assinatura do método:

BSTR getExtensaoOID(int posicao);

Parâmetros:

posicao	int	posição da lista de Extensões iniciando da posição 0

Retorno:

retorno	BSTR	**Sucesso:**valor do OID da extensão na posição se existir **Erro:**vazio

MÉTODO: getExtensaoCritica

Esta função tem o objetivo de informar se a extensão na posição, passada como parâmetro, na lista de extensões do certificado é uma extensão critica.

Assinatura do método:

Int getExtensaoCritica(int posicao);

Parâmetros:

posicao	int	posição da lista de Extensões iniciando da posição 0

Retorno:

retorno	int	**Crítica:** 1 **Não Crítica:** 0

MÉTODO: getCountCertificadosCadeia

Recupera o número de certificados existentes na cadeia de certificação do certificado em questão. Esta função deve ser chamada após a execução da função verificarCadeiaCertificacao.

Assinatura do método:

int getCountCertificadosCadeia();

Parâmetros: Esta função não tem parâmetros.

Retorno:

retorno	int	número de certificados na cadeia de certificação do certificado

MÉTODO: getStatusCadeiaCertificacao

Recupera o status do certificado na posição passada como parâmetro da lista de certificados na cadeia de certificação do certificado. Esta função deve ser chamada após a execução da função verificarCadeiaCertificacao. O certificado na posição 0 é o certificado final (do usuário), o que está sendo construída a cadeia de certificação, se existir, na posição 1 teremos o status do emissor deste certificado, e assim por diante até o certificado raiz.

Assinatura do método:

int getStatusCadeiaCertificacao(int __posicao);

Parâmetros:

__posição	int	posição do certificado na lista de certificados da cadeia de certificação iniciando da posição 0

Retorno:

Retorno	int	valor do status do certificado (**)

(**) Os valores para Status podem ser:

CERT_TRUST_HAS_EXACT_MATCH_ISSUER	0x00000001
CERT_TRUST_HAS_KEY_MATCH_ISSUER	0x00000002
CERT_TRUST_HAS_NAME_MATCH_ISSUER	0x00000004
CERT_TRUST_IS_SELF_SIGNED	0x00000008
CERT_TRUST_HAS_PREFERRED_ISSUER	0x00000100
CERT_TRUST_HAS_ISSUANCE_CHAIN_POLICY	0x00000200
CERT_TRUST_HAS_VALID_NAME_CONSTRAINTS	0x00000400

Portanto para um retorno = 9, temos que o certificado é auto-assinado e o emissor é um certificado confiável.

MÉTODO: getErrorCadeiaCertificacao

Recupera o erro do certificado na posição passada como parâmetro da lista de certificados na cadeia de certificação do certificado. Esta função deve ser chamada após a execução da função verificarCadeiaCertificacao. O certificado na posição 0 é o certificado final (do usuário), o que está sendo construída a cadeia de certificação, se existir, na posição 1 teremos o erro do emissor deste certificado, e assim por diante até o certificado raiz.

Assinatura do método:

int getErrorCadeiaCertificacao(int __posicao);

Parâmetros:

__posicao	int	posição do certificado na lista de certificados da cadeia de certificação iniciando da posição 0

Retorno:

Retorno	int	valor do erro do certificado (**)

(**) Os valores para erro podem ser:

CERT_TRUST_NO_ERROR	0x00000000
CERT_TRUST_IS_NOT_TIME_VALID	0x00000001
CERT_TRUST_IS_NOT_TIME_NESTED	0x00000002
CERT_TRUST_IS_REVOKED	0x00000004
CERT_TRUST_IS_NOT_SIGNATURE_VALID	0x00000008
CERT_TRUST_IS_NOT_VALID_FOR_USAGE	0x00000010
CERT_TRUST_IS_UNTRUSTED_ROOT	0x00000020
CERT_TRUST_REVOCATION_STATUS_UNKNOWN	0x00000040
CERT_TRUST_IS_CYCLIC	0x00000080
CERT_TRUST_INVALID_EXTENSION	0x00000100
CERT_TRUST_INVALID_POLICY_CONSTRAINTS	0x00000200
CERT_TRUST_INVALID_BASIC_CONSTRAINTS	0x00000400
CERT_TRUST_INVALID_NAME_CONSTRAINTS	0x00000800
CERT_TRUST_HAS_NOT_SUPPORTED_NAME_CONSTRAINT	0x00001000
CERT_TRUST_HAS_NOT_DEFINED_NAME_CONSTRAINT	0x00002000
CERT_TRUST_HAS_NOT_PERMITTED_NAME_CONSTRAINT	0x00004000
CERT_TRUST_HAS_EXCLUDED_NAME_CONSTRAINT	0x00008000
CERT_TRUST_IS_OFFLINE_REVOCATION	0x01000000
CERT_TRUST_NO_ISSUANCE_CHAIN_POLICY	0x02000000

Portanto para um retorno = 9, temos que o certificado não tem uma assinatura válida e o tempo do certificado não é valido, ou seja, o certificado expirou ou ainda não é valido.

MÉTODO: getHashCadeiaCertificacao

Recupera o "hash" do certificado na posição passada como parâmetro da lista de certificados na cadeia de certificação do certificado. Esta função deve ser chamada após a execução da função verificarCadeiaCertificacao. O certificado na posição 0 é o certificado final (do usuário), o que está sendo construída a cadeia de certificação, se existir, na posição 1 teremos o "hash" do emissor deste certificado, e assim por diante até o certificado raiz.

Assinatura do método:

BSTR getHashCadeiaCertificacao(int __posicao);

Parâmetros:

__posição	int	posição do certificado na lista de certificados da cadeia de certificação iniciando da posição 0

Retorno:

Retorno	BSTR	valor do hash SHA-1 em Hexadecimal do certificado

MÉTODO: verificarCadeiaCertificacao

Nesta função é inicializada uma lista contendo todos os certificados da cadeia de certificação do certificado que está chamando a função. Cada objeto da lista de certificados da cadeia possui seu hash, e um valor para status e erro. O número de certificados da lista pode ser adquirido com getCountCertificadosCadeia.

Assinatura do método:

int verificarCadeiaCertificacao();

Parâmetros: Esta função não tem parâmetros.

Retorno:

```
retorno         int             (**)
```

(**) Os valores de retorno podem ser:

```
CERT_TRUST_NO_ERROR                             0x00000000
CERT_TRUST_IS_NOT_TIME_VALID                    0x00000001
CERT_TRUST_IS_NOT_TIME_NESTED                   0x00000002
CERT_TRUST_IS_REVOKED                           0x00000004
CERT_TRUST_IS_NOT_SIGNATURE_VALID               0x00000008
CERT_TRUST_IS_NOT_VALID_FOR_USAGE               0x00000010
CERT_TRUST_IS_UNTRUSTED_ROOT                    0x00000020
CERT_TRUST_REVOCATION_STATUS_UNKNOWN            0x00000040
CERT_TRUST_IS_CYCLIC                            0x00000080
CERT_TRUST_INVALID_EXTENSION                    0x00000100
CERT_TRUST_INVALID_POLICY_CONSTRAINTS           0x00000200
CERT_TRUST_INVALID_BASIC_CONSTRAINTS            0x00000400
CERT_TRUST_INVALID_NAME_CONSTRAINTS             0x00000800
CERT_TRUST_HAS_NOT_SUPPORTED_NAME_CONSTRAINT    0x00001000
CERT_TRUST_HAS_NOT_DEFINED_NAME_CONSTRAINT      0x00002000
CERT_TRUST_HAS_NOT_PERMITTED_NAME_CONSTRAINT    0x00004000
CERT_TRUST_HAS_EXCLUDED_NAME_CONSTRAINT         0x00008000
CERT_TRUST_IS_OFFLINE_REVOCATION                0x01000000
CERT_TRUST_NO_ISSUANCE_CHAIN_POLICY             0x02000000

CERT_TRUST_IS_PARTIAL_CHAIN                     0x00010000
CERT_TRUST_CTL_IS_NOT_TIME_VALID                0x00020000
CERT_TRUST_CTL_IS_NOT_SIGNATURE_VALID           0x00040000
CERT_TRUST_CTL_IS_NOT_VALID_FOR_USAGE           0x00080000
```

Portanto para um retorno = 9, temos que algum certificado da cadeia de certificação não tem uma assinatura válida e o tempo do certificado não é valido,

ou seja, o certificado expirou ou ainda não é válido para algum dos certificados da cadeia.

Para saber qual dos certificados da cadeia que não tem uma assinatura válida, podemos verificar o erro dos certificados um a um através do método verificarCadeiaCertificacao.

MÉTODO: existeChavePrivada

O objetivo desta função é verificar se existe uma chave privada associada ao certificado que está no repositório de certificados do Windows.
ATENÇÃO: Este método deve ser utilizado somente para certificados do tipo A1. NÃO DEVE ser utilizado para certificados A3 (armazenados em token e/ou smart card) pois o retorno será sempre 0 (não existe).

Assinatura do método:

BOOL existeChavePrivada();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BOOL	**Existe:** 1 **Não existe:**0

MÉTODO: setDataVerifica

O objetivo desta função é adicionar uma data que será utilizada para a verificação da validade do certificado. Normalmente está função é utilizada para setar uma data do carimbo do tempo.

Assinatura do método:

int setDataVerifica (BSTR __data);

Parâmetros:

__data	BSTR	String contendo a data no seguinte formato: dd/mm/aaaa.

Retorno:

Retorno	int	**Sucesso:** 1 **Erro:** verificar valores de retorno dos erros na tabela de erros

MÉTODO: getUsoExtendidoChaves

Recupera o Uso Estendido na posição, passada como parâmetro, na lista de Usos Estendidos do certificado.

Assinatura do método:

int getUsoExtendidoChaves (int __posicao);

Parâmetros:

__posicao	int	posição do uso estendido na lista de usos estendidos do certificado iniciando da posição 0

Retorno:

Retorno	BSTR	**Sucesso:** OID do Uso Estendido **Erro:** vazio

MÉTODO: getCountUsoExtendidoChaves

Recupera o número de Usos Estendidos do certificado em questão.

Assinatura do método:

int getCountUsoExtendidoChaves ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número de elementos da lista **Erro:** 0

MÉTODO: isAC

Verifica se o certificado é do tipo autoridade certificadora

Assinatura do método:

int isAC ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** 1 caso autoridade certificadora, 0 caso não autoridade certificadora **Erro:** verificar valores de retorno dos erros na tabela de erros.

MÉTODO: isRestricaoCaminho

Verifica se o certificado possui restrições quanto ao caminho de certificação

Assinatura do método:

int isRestricaoCaminho ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** 1 caso possua, 0 caso não possua. **Erro:** verificar valores de retorno dos erros na tabela de erros.

MÉTODO: getTamanhoRestricaoCaminho

Retorna o tamanho da restrição quanto ao caminho de certificação

Assinatura do método:

int getTamanhoRestricaoCaminho ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** número contendo o tamanho. **Erro:** verificar valores de retorno dos erros na tabela de erros.

MÉTODO: getTipoICPBrasil

Retorna qual o tipo de certificado ICP-Brasil, A1, A2, A3, A4 ou S1, S2, S3, S4.

Assinatura do método:

int getTipoICPBrasil ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno int **Sucesso:** Algum dos números segundo a tabela abaixo.

1	A1
2	A2
3	A3
4	A4
101	S1
102	S2
103	S3
104	S4

Erro: verificar valores de retorno dos erros na tabela de erros.

MÉTODO: getHashIdentificadorAutoridade

Retorna o hash identificador da chave da autoridade da extensão Identificador da Chave da Autoridade.

Assinatura do método:

BSTR getHashIdentificadorAutoridade ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno BSTR **Sucesso:** String contando o hash sha1 do identificador da chave ad autoridade.
Erro: vazios.

MÉTODO: getCountAssuntoOU

Retorna a quantidade de assuntoOU do certificado.

Assinatura do método:

int getCountAssuntoOU();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número contando a quantidade de assuntoOU da lista. **Erro:** 0.

MÉTODO: getMultiplosAssuntoOU

Retorna o assuntoOU do certificado.

Assinatura do método:

BSTR getMultiplosAssuntoOU (int __posicao);

Parâmetros:

__posicao	int	posição do AssuntoOU na lista do certificado iniciando da posição 0

Retorno:

Retorno	BSTR	**Sucesso:** String contendo o assuntoOU de acordo com a posição passada como parâmetro. **Erro:** vazio.

MÉTODO: getCountEmissorOU

Retorna a quantidade de emissorOU do certificado.

Assinatura do método:

int getCountEmissorOU ();

Parâmetros: Está função não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número contando a quantidade de emissorOU da lista. **Erro:** 0.

MÉTODO: getMultiplosEmissorOU

Retorna o emissorOU do certificado.

Assinatura do método:

BSTR getMultiplosEmissorOU (int __posicao);

Parâmetros:

__posicao	int	posição do EmissorOU na lista do certificado iniciando da posição 0

Retorno:

Retorno	BSTR	**Sucesso:** String contendo o EmissorOU de acordo com a posição passada como parâmetro. **Erro:** 0.

MÉTODO: setCRL

Adicionar um objeto do tipo CRL para verificar o certificado. Utilizado caso deseja verificar o certificado com uma crl externa.

Assinatura do método:

int setCRL (ICRL __crl);

Parâmetros:

__crl	ICRL	Objeto do tipo ICRL.

Retorno:

Retorno	int	**Sucesso:** 1 caso a crl seja aceita para verificar o certificado. **Erro:** verificar valores de retorno dos erros na tabela de erros.

MÉTODO: getLCR

Esta função tem o objetivo de retornar um objeto do tipo CRL. Este método somente deve ser utilizado após definir um CRL através do método setCRL(ICRL __crl) .

Assinatura do método:

ICRL getLCR ();

Parâmetros: Não possui paramêtros

Retorno:

Retorno	ICRL	**Sucesso:** Objeto do tipo ICRL. **Erro:** vazio.

MÉTODO: getMotivoRevogacao

Retorna o motivo pelo qual o certificado foi revogado. Está função somente retorna um valor após a execução dos métodos ñstatusò ou ñverificarCRLò e caso as mesmas tenham retornado que o certificado está revogado.

Assinatura do método:

BSTR getMotivoRevogacao ();

Parâmetros: Não possui paramêtros

Retorno:

Retorno BSTR **Sucesso:** String contendo o motivo de acordo com a tabela abaixo.

0	Não especificado
1	Compromisso da chave
2	Compromisso da Autoridade Certificadora
3	Mudança de filiação
4	Certificado substituído
5	Cessação de operação
6	Certificado suspenso
8	Removido da CRL
9	Remoção de privilégio
10	Remoção de privilégio

MÉTODO: getDataRevogacao

Retorna a data que o certificado foi revogado de acordo com a data da CRL. Está função somente retorna um valor após a execução dos métodos ñstatusò ou ñverificarCRLò e caso as mesmas tenham retornado que o certificado está revogado.

Assinatura do método:

BSTR getDataRevogacao ();

Parâmetros: Não possui paramêtros

Retorno:

Retorno	BSTR	**Sucesso:** String contendo a data de revogação no formato dd/mm/aaaa. **Erro:** vazio.

MÉTODO: isAutoAssinada

Retorna se o certificado é auto-assinado.

Assinatura do método:

int isAutoAssinada();

Parâmetros: Não possui paramêtros

Retorno:

Retorno	int	**Sucesso:** 1 caso seja auto-assinado, 0 caso contrário. **Erro:** vazio.

MÉTODO: getTamanhoChave

Retorna o tamanho da chave pública correspondente ao certificado em bits.

Assinatura do método:

int getTamanhoChave();

Parâmetros: Não possui paramêtros

Retorno:

Retorno	int	**Sucesso:** Tamanho da chave pública do certificado em bits. **Erro:** 0.

4.3 Classe Assinador (ClassID: D74AA60A-D77B-4482-8850-4768E636AAC7)

A classe Assinador foi desenvolvida com a tecnologia COM. Esta classe exporta os métodos de assinatura, co-assinatura e cálculo de hash.

Os métodos implementados nesta classe estão descritos a seguir:

MÉTODO assineArquivo

Este método assina digitalmente um arquivo que encontra-se em qualquer formato, adicionando o nome do arquivo assinado, a data e hora da assinatura e se desejado uma descrição. O formato do arquivo assinado segue o padrão PKCS#7.

O retorno da função é um inteiro, com o status do processo. Se ocorrer erro o status é um valor negativo.

O nome do arquivo assinado resultante é o passado no parâmetro caminho acrescido da extensão .p7s, por exemplo, ao assinar o arquivo teste.txt, passando no parâmetro caminho o valor c:\teste.txt, será criado o arquivo assinado c:\teste.txt.p7s, se o parâmetro caminho não informar o diretório e o nome do arquivo, o SDK utilizará o mesmo valor do arquivo que está sendo assinado. Caso o arquivo já exista, o mesmo será sobreescrito.

Assinatura do método:

int AssineArquivo (BSTR __arq, BSTR __caminho, BSTR __descricao, BSTR __hashCert);

Parâmetros:

__arq	BSTR	Caminho do arquivo a ser assinado
__caminho	BSTR	Caminho onde será armazenado o arquivo assinado, por exemplo: "C:\Windows\arqAssinado". A extensão é colocada automaticamente.
__descricao	BSTR	Descrição do arquivo assinado.
__hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.

Retorno:

retorno	int	**Sucesso:**1 **Erros comuns:** CERTIFICADO_INVALIDO_ASSINATURA CERTICADO_NAO_ENCONTRADO ARQUIVO_INEXISTENTE

		ARQUIVO_VAZIO ERRO_INTERNO_CRYPTO ERRO_MEMORIA DESTINO_INVALIDO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO assineMem

Este método assina digitalmente um texto em memória, que se encontra em uma variável do tipo *string (caso o conteúdo a ser assinado for binário, deve-se codificá-lo em hexadecimal ou base64)*, adicionando a data e hora da assinatura e se desejado uma descrição. O formato do bloco de memória assinado segue o padrão PKCS#7 e estará codificado em Hexadecimal ou Base64 de acordo com o valor setado através da função setFormatoDadosMemória.

O retorno da função é um inteiro, com o status do processo. A assinatura é retornada no parâmetro __assinatura, no formato setado. Se ocorrer erro o status é um valor negativo.

Para salvar a assinatura em um arquivo, deve-se convertê-la para binário.

Se o código que está sendo gerado for para execução em navegadores de internet, leia a documentação da função retorneAssinaturaMem.

Assinatura do método:

int assineMem (BSTR __mem, BSTR __descricao, BSTR __hashCert, BSTR* __assinatura);

Parâmetros:

__mem	BSTR	Texto a ser Assinado (deve ser string)
__descricao	BSTR	Descrição do arquivo assinado.
__hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.
__assinatura	BSTR*	Assinatura da memória (em hexadecimal).

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** CERTIFICADO_INVALIDO_ASSINATURA CERTICADO_NAO_ENCONTRADO ERRO_INTERNO_CRYPTO ERRO_MEMORIA ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO coAssineArquivo

Este método adiciona uma assinatura em um arquivo já assinado (Co-Assinatura), adicionando o nome do arquivo, a data e hora da assinatura e se desejado uma descrição.

O formato do arquivo co-assinado segue está estrutura (pkcs#7):

```
signed Data{
      version
      digest algorithms
      data
      certificates{
        first signer certificate
        co signer certificate
        }//end certificates
        signer info{
        first signer Info
        co-signer info
      }//end signer info
}//end signed data
```

Caso o arquivo passado como parâmetro não seja um arquivo já assinado, esta função detectará automaticamente isto e ao invés de co-assinar o arquivo irá efetuar a assinatura do mesmo, pois o mesmo ainda não continha nenhuma assinatura.

O retorno da função é um inteiro, com o status do processo. Se ocorrer erro o status é um valor negativo.

O nome do arquivo co-assinado é o passado no parâmetro caminho acrescido da extensão .p7s, por exemplo, ao co-assinar o arquivo teste.txt.p7s, passando no parâmetro caminho o valor c:\teste.txt, será criado o arquivo assinado c:\teste.txt.p7s, se o parâmetro caminho não informar o diretório e o nome do arquivo, o SDK utilizará o mesmo valor do arquivo que está sendo co-assinado. Caso o arquivo já exista, o mesmo será sobreescrito.

Assinatura do método:

int coAssineArquivo (BSTR __arq, BSTR __caminho, BSTR __descricao, BSTR __hashCert);

Parâmetros:

__arq	BSTR	Caminho do arquivo a ser coassinado (.p7s)
__caminho	BSTR	Caminho onde será armazenado o arquivo assinado, por exemplo: "C:\Windows\arqAssinado". A extensão é colocada automaticamente.
__descricao	BSTR	Descrição do arquivo assinado.

__ hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.

Retorno:

Retorno	BSTR	**Sucesso:** 1 **Erros comuns:** CERTIFICADO_INVALIDO_ASSINATURA CERTICADO_NAO_ENCONTRADO ARQUIVO_INEXISTENTE ARQUIVO_VAZIO ERRO_INTERNO_CRYPTO ERRO_MEMORIA DESTINO_INVALIDO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO coAssineMem

Este método co-assina digitalmente um texto em memória, que se encontra em uma variável do tipo *string (caso o conteúdo a ser assinado for binário, deve-se codificá-lo em hexadecimal ou base64)*, adicionando a data e hora da assinatura e se desejado uma descrição. O formato do bloco de memória co-assinado segue o padrão PKCS#7 e estará codificado em Hexadecimal ou Base64 de acordo com o valor setado através da função setFormatoDadosMemória.

Este método irá adicionar uma assinatura em um bloco de memória já assinado (Co-Assinatura)

O formato do bloco de memória co-assinado segue está estrutura (pkcs#7):

```
signed Data{
      version
      digest algorithms
      data
      certificates{
        first signer certificate
        co signer certificate
        }//end certificates
        signer info{
        first signer Info
        co-signer info
      }//end signer info
}//end signed data
```

Caso o bloco de memória passado como parâmetro ainda não esteja assinado, esta função detectará automaticamente isto e ao invés de co-assinar o bloco de memória irá efetuar a assinatura do mesmo, pois o mesmo ainda não continha nenhuma assinatura.

É importante informar que se o formato dos dados passados como parâmetro for diferente do formato setado pela função setFormatoDadosMemoria, ocorrerá um erro na conversão de tipos, e isto resultará na assinatura dos dados passados como parâmetro, pois a função reconhecerá que como houve um erro de conversão, os dados são dados ainda não assinados.

O retorno da função é um inteiro, com o status do processo. A co-assinatura é retornada no parâmetro __coAssinatura, no formato setado. Se ocorrer erro o status é um valor negativo.

Para salvar a co-assinatura em um arquivo, deve-se convertê-la para binário.

Se o código que está sendo gerado for para execução em navegadores de internet, leia a documentação da função retorneAssinaturaMem.

Assinatura do método:

int coAssineMem (BSTR __mem, BSTR __descricao, BSTR __hashCert, BSTR* __coAssinatura);

Parâmetros:

__mem	BSTR	Texto a ser Assinado (deve ser string)
__descricao	BSTR	Descrição do arquivo assinado.
__hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.
__coAssinatura	BSTR*	Assinatura da memória (em hexadecimal).

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** CERTIFICADO_INVALIDO_ASSINATURA CERTICADO_NAO_ENCONTRADO ARQUIVO_INEXISTENTE ARQUIVO_VAZIO ERRO_INTERNO_CRYPTO ERRO_MEMORIA ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO: retorneAssinaturaMem

Esta função tem o objetivo de recuperar a assinatura, no formato setado pela função setFormatoDadosMemoria, após a execução da função assineMem.

Esta função somente é necessária quando a classe está sendo executada por navegadores de internet, pois a assinatura não é retornada no parâmetro __assinatura da função assineMem quando executada nos navegadores.

Para salvar a assinatura em um arquivo, deve-se convertê-la para binário.

Assinatura do método:

BSTR retorneAssinaturaMem ();
Parâmetros: Esta função não tem parâmetros

Retorno:

retorno	BSTR	Assinatura da memória (no formato setado).

MÉTODO: setFormatoDadosMemoria

Seta o formato de codificação/decodificação que será utilizado na classe assinador. Este formato será utilizado na hora de codificar o conteúdo assinado em memória, codificar o conteúdo co-assinado em memória e decodificar o parâmetro de dados a serem co-assinados, pois os mesmos deverão ter sidos gerados pelo método assineMem.

Se esta função nunca for utilizada, o valor default utilizado será a codificação/decodificação em HEXADECIMAL.

Quando se deseja setar qual o formato de codificação/decodificação que será utilizado, deve-se chamar este método antes dos métodos que realizam a assinatura.

Assinatura do método:

int setFormatoDadosMemoria(int __formato);

Parâmetros:

__formato	int	Formato a ser utilizado pela classe. Os valores podem ser: 0 = HEXADECIMAL (default) 1 = BASE64

Retorno:

retorno	int	**Sucesso:** 1 **Erro:** ERRO_FORMATO_INVALIDO

MÉTODO: setDataHora

Seta o valor de data/hora que deverá ser incluído numa assinatura/co-assinatura, a data/hora informada será tratada como padrão UTC (sem fuso horário).

Se nenhum valor for setado, a data/hora utilizada será a do sistema operacional onde a aplicação estiver sendo executada.

Quando se deseja setar qual a data/hora que deverá ser adicionada na assinatura ou co-assinatura, este método sempre deverá ter sido chamado antes dos métodos que realizam a assinatura. Se esta função for chamada uma vez, e em seguida chamarmos duas vezes a função assinarArquivo, a data/hora do primeiro arquivo assinado será a passada como parâmetro na função setDataHora e a data/hora da segunda assinatura será a do sistema operacional, portanto se o desejado era setar a data/hora nas duas assinaturas, deve-se chamar o método setDataHora, depois o de assinatura, depois novamente setDataHora e por fim novamente o de assinatura.

Assinatura do método:

int setDataHora (int __dia, int __mes, int __ano, int __hora, int __minutos, int __segundos, int __milisegundos);

Parâmetros:

__dia	int	Dia a ser setado
__mes	int	Mês a ser setado
__ano	int	Ano a ser setado (entre 1950 e 2049)
__hora	int	Hora a ser setada
__minutos	int	Minuto a ser setado
__segundos	int	Segundo a ser setado
__milisegundos	int	Milisegundo a ser setado

Retorno:

retorno	int	**Sucesso:** 1 **Erro:** ERRO_DATA_INVALIDA

MÉTODO: setAddCrlCaminhoCert

Adicionada ao assinar um documento as CRLs e o caminho de certificação do certificado do assinante. Ex: Se Fulano assinou o arquivo ñteste.txtò no momento da assinatura será adicionado as CRLs mais atuais e o caminho de certificação do Fulano.

Obs: As CRLs e o caminho de certificação devem estar instalados na máquina no momento da assinatura.

Assinatura do método:

int setAddCrlCaminhoCert (int __adicionar);

Parâmetros:

__formato	int	Os valores podem ser:

		0 = para não adicionar (defualt) 1 = para adicionar.

Retorno:

retorno	int	**Sucesso:** 1

MÉTODO: hashDados

Calcula o hash SHA-1 dos dados passados como parâmetro.

Assinatura do método:

BSTR hashDados (BSTR __dados, int __tamanho);

Parâmetros:

__dados	BSTR	Dados que terão o hash calculados
__tamanho	int	Tamanho dos dados do primeiro parâmetro

Retorno:

retorno	BSTR	**Sucesso:** hash SHA-1 em hexadecimal **Erro:** vazio

MÉTODO: hashDoArquivo

Calcula o hash SHA-1 do arquivo passado como parâmetro.

Assinatura do método:

BSTR hashDoArquivo (BSTR __arquivo);

Parâmetros:

__arquivo	BSTR	Caminho completo do arquivo que terá o hash calculado

Retorno:

retorno	BSTR	**Sucesso:** hash SHA-1 em hexadecimal **Erro:** vazio

MÉTODO assineArquivoDetached

Este método assina digitalmente um arquivo no formato detached. Na assinatura detached o dado assinado não faz parte da assinatura.

O retorno da função é um inteiro, com o status do processo. Se ocorrer erro o status é um valor negativo.

Assinatura do método:

assineArquivoDetached(BSTR __arq, BSTR __caminho, BSTR __descricao, BSTR __hashCert, int *__status)

Parâmetros:

arq	BSTR	Arquivo a ser Assinado (deve ser string)
caminho		Caminho do arquivo a ser assinado
descricao	BSTR	Descrição do arquivo assinado.
hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.
status	int	

Retorno:

status	int	**Sucesso:** 1 **Erros comuns:** números negativos ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO assineMemDetached

Este método assina digitalmente um texto em memória, que se encontra em uma variável do tipo *string*, adicionando a data e hora da assinatura e se desejado uma descrição. Na assinatura detached o dado assinado não faz parte da assinatura.

O retorno da função é um inteiro, com o status do processo. A assinatura é retornada no parâmetro __assinatura, no formato setado. Se ocorrer erro o status é um valor negativo.

Assinatura do método:

assineMemDetached(BSTR __mem, BSTR __descricao, BSTR __hashCert, BSTR *__assinatura, int *__status)

Parâmetros:

mem	BSTR	Texto a ser Assinado (deve ser string)
descricao	BSTR	Descrição do arquivo assinado.
hashCert	BSTR	Hash sha-1 em Hexadecimal do certificado digital usado para assinatura. Não deve possuir espaços entre os valores do hash.
assinatura	BSTR*	Assinatura da memória (em hexadecimal).
status	int	

Retorno:

status	int	**Sucesso:** 1 **Erros comuns:** números negativos ... *verificar valores de retorno dos erros na tabela de erros

4.4 Classe Verificador (ClassID: 1690EE68-8D31-473d-9CEB-C51D56A76D7B)

A classe Verificador exporta os métodos da verificação e status da verificação. Quando um arquivo assinado é verificado, o status da verificação fica disponível através dos métodos getCertificado, getStatusAssinatura, getStatusCertificado, getNomeArquivo, getDescricao e getArquivo . O certificado retornado é da classe Certificado, possibilitando a extração de todos os dados do certificado.

Os métodos da classe Verificador estão descritos abaixo:

MÉTODO verifiqueAssinatura

Este método verifica a assinatura digital de arquivo assinado, no formato p7s, recuperando os dados da assinatura.

Para cada verificação é gerado um objeto contendo: certificado digital utilizado na assinatura, nome do arquivo original, descrição, *status* de verificação e status do certificado.

Se todos os dados foram recuperados esta função retornará SUCESSO, no entanto isto não significa que a assinatura, ou certificados digital são válidos.

Assinatura do método:

int verifiqueAssinatura (BSTR __arq);

Parâmetros:

__arq	BSTR	Caminho do arquivo a ser verificado

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** ARQUIVO_INEXISTENTE ARQUIVO_VAZIO HASH_DIFERENTES ERRO_ASSINATURA ERRO_INTERNO_CRYPTO ARQUIVO_INVALIDO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO VerifiqueAssinaturaMem

Este método verifica a assinatura digital de um bloco de memória, codificado no formato indicado pelo método setFormatoDadosMemoria da classe Verificador, recuperando os dados da assinatura.

Para cada verificação é gerado um objeto contendo: certificado digital utilizado na assinatura, descrição, ***status*** de verificação e status do certificado.

Lembre-se de antes de verificar uma assinatura em memória, setar o formato de codificação apropriado que a classe verificador deverá utilizar.

Assinatura do método:

int verifiqueAssinaturaMem (BSTR __mem);

Parâmetros:

__mem	BSTR	Memória a ser verificada

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** ERRO_CONTEUDO_VAZIO ERRO_HEX_BIN ERRO_DECODIFICANDO_BASE_64 ERRO_MEMORIA HASH_DIFERENTES ERRO_ASSINATURA ERRO_INTERNO_CRYPTO ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO ExtrairDocumento

Verifica se a assinatura é válida, e extrai o arquivo/documento contido no arquivo assinado (p7s). Temos a opção de abri-lo ou salvá-lo.

O retorno da função é um inteiro, com o status do processo. Se ocorrer erro o status é um valor negativo.

Se a opção de verificar a assinatura for ativada, o procedimento de abrir ou salvar somente será realizado se todas as assinaturas estiverem corretas. Os certificados e cadeia de certificação não são verificados nesta função.

Assinatura do método:

Int extrairDocumento (BSTR __arq, BSTR __caminho, int __flag, bool __verificar, bool __sobrescrever)

Parâmetros:

__arq	BSTR	Caminho do arquivo assinado.
__caminho	BSTR	Caminho onde será salvo o documento extraído.
__flag	Int	indica se é para ABRIR(1), SALVAR(2) documento.
__verificar	Bool	Indica se deve verificar a assinatura (se o arquivo esta assinado corretamente ou não) antes de abrir ou salvar um arquivo. True=verificar e false abre o arquivo mesmo que a assinatura esteja inválida.
__sobrescrever	Bool	Indica se devemos sobre-escrever o documento, caso ele exista, ou não

Retorno:

retorno	Int	**Sucesso:** 1 **Erros comuns:** ARQUIVO_INEXISTENTE ARQUIVO_INVALIDO ERRO_INTERNO_CRYPTO ASSINATURA_SEM_NOME_ARQUIVO ARQUIVO_JA_EXISTE ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO getCountAssinaturas

Retorna a quantidade de assinaturas de um determinado arquivo assinado. Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

Int getCountAssinaturas();

Parâmetros: Esta função não tem parâmetros.

Retorno:

retorno	int	Número de assinaturas.

MÉTODO getStatusCertificado

Retorna o status do certificado utilizado na Assinatura, se ele está na CRL ou não e se estiver dentro do prazo de validade (neste caso utilizada a hora da máquina). O status será o referente ao certificado da assinatura correspondente ao índice passado como parâmetro.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Obs: Caso exista um Carimbo de Tempo na assinatura, na verificação do certificado da assinatura será levado em conta à hora do carimbo. Para validação do estado de revogação e necessário que as LCRs estejam dentro do arquivo assinado e que as mesmas sejam válidas no instante de tempo do carimbo de tempo. As LCRs atuais instaladas no computador não serão utilizadas.

Assinatura do método:

Int getStatusCertificado(int __index);

Parâmetros:

__index	int	Índice do status do certificado desejado iniciando da posição 0.

Retorno:

retorno	int	Status do Certificado (ver função Status da classe Certificado)

MÉTODO getStatusAssinatura

Retorna o status da verificação da assinatura (verifica a integridade do documento). O status será o referente à assinatura correspondente ao índice passado como parâmetro.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

Int getStatusAssinatura(int __index);

Parâmetros:

__index	int	Índice da verificação desejada iniciando da posição 0.

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** ARQUIVO_INEXISTENTE ARQUIVO_INVALIDO ERRO_INTERNO_CRYPTO ASSINATURA_SEM_NOME_ARQUIVO ARQUIVO_JA_EXISTE ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO getStatusCarimboTempo

Caso a assinatura possua um atributo não autenticado de carimbo de tempo retorna o status da verificação deste carimbo (verifica a integridade do carimbo, se o hash do carimbo é igual ao da assinatura e se sua codificação está correta). O status do carimbo será o referente à assinatura que detém o mesmo correspondente ao índice passado como parâmetro.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

Int getStatusCarimboTempo (int __index);

Parâmetros:

__index	int	Índice da verificação desejada iniciando da posição 0.

Retorno:

retorno	int	**Sucesso:** 1 **Erros comuns:** NAO_EXISTE_TIME_STAMP ERRO_ESTRUTURA_TSTINFO ERRO_DECODIFICANDO_DATA_TIME_STAMP DATA_ARQUIVO_NAO_ENCONTRADO ERRO_INTERNO_CRYPTO ERRO_ASSINATURA_TST_INVALIDA ERRO_VERIFICANDO_TIME_STAMP ... *verificar valores de retorno dos erros na tabela de erros

MÉTODO getArquivo

Retorna o arquivo que foi assinado (ao invés de gravar o arquivo através do método ñExtrairDocumentoAssinadoò você pode verificar em memória o seu conteúdo). No entanto o retorno será uma string do conteúdo assinado. Este método portanto apenas poderá ser utilizado quando se assina documentos de texto, já quando assinamos arquivos binários, este método retornará uma string com o valor do arquivo acarretando em caracteres inválidos e truncamento da informação.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

BSTR getArquivo(int __index);

Parâmetros:

__index	int	Deve ser sempre 0, pois a informação é carregada apenas para o primeiro assinante.

Retorno:

retorno	BSTR	**Sucesso:** Conteúdo assinado **Erro:** vazio

MÉTODO getNomeArquivo

Retorna o nome do arquivo que foi assinado.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò

Assinatura do método:

BSTR getNomeArquivo(int __index);

Parâmetros:

__index	int	Índice do nome do arquivo desejado iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:**Nome do arquivo **Erro:** vazio

MÉTODO getDescricao

Retorna a descrição da assinatura. A descrição será a referente à assinatura correspondente ao índice passado como parâmetro.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

BSTR getDescricao(int __index);

Parâmetros:

__index	Int	Índice da descrição da assinatura iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** Descrição da assinatura **Erro:** vazio

MÉTODO getCertificado

Retorna o certificado utilizado na assinatura. O certificado será o referente à assinatura correspondente ao índice passado como parâmetro.

Com o objeto retornado nesta função é possível acessar qualquer método da classe Certificados.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

Icertificado getCertificado(int __index);

Parâmetros:

__index	int	Índice do certificado da assinatura iniciando da posição 0.

Retorno:

retorno	ICertificado	**Sucesso:** Objeto da classe Certificado **Erro:** NULL

MÉTODO getCarimboAssinatura

Retorna um objeto Carimbo que representa o carimbo de tempo (Atributo não autenticado) da assinatura. O objeto Carimbo será o referente à assinatura correspondente ao índice passado como parâmetro.

Com o objeto retornado nesta função é possível acessar qualquer método da classe Carimbo.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

ICarimbo getCarimboAssinatura (int __index);

Parâmetros:

__index	int	Índice do carimbo da assinatura iniciando da posição 0.

Retorno:

retorno	ICarimbo	**Sucesso:** Objeto da classe Carimbo **Erro:** NULL

MÉTODO getDataCarimboTempo

Retorna a data do carimbo de tempo da assinatura. Não retorna a data UTC, retorna a data convertida para o fuso local.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

BSTR getDataCarimboTempo (int __index);

Parâmetros:

__index	int	Índice da assinatura iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** String contendo a data do carimbo de tempo da assinatura **Erro:** ""

MÉTODO getHash

Retorna o "hash" do certificado utilizado na assinatura. O hash do certificado será o referente à assinatura correspondente ao índice passado como parâmetro.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos "verifiqueAssinatura" ou "verifiqueAssinaturaMem"

Assinatura do método:

BSTR getHash(int __index);

Parâmetros:

__index	int	Índice do certificado da assinatura iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** Hash Sha-1 do Certificado, em hexadecimal, utilizado na assinatura **Erro:** vazio

MÉTODO getStatusCadeia

Retorna o status da cadeia de certificação do certificado de acordo com a função getErrorCadeiaCertificacao. O Status é o referente à assinatura correspondente ao índice passado como parâmetro

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos "verifiqueAssinatura" ou "verifiqueAssinaturaMem"

Obs: Caso exista um Carimbo de Tempo na assinatura, na verificação da cadeia do certificado da assinatura será levado em conta à hora do carimbo. Para validação do estado de revogação e necessário que as LCRs estejam dentro do arquivo assinado e que as mesmas sejam válidas no instante de tempo do carimbo de tempo. As LCRs atuais instaladas no computador não serão utilizadas.

Assinatura do método:

Int getStatusCadeia(int __index);

Parâmetros:

__index	Int	Índice do certificado da assinatura iniciando da posição 0.

Retorno:

Retorno	int	Consultar retornos da função getErrorCadeiaCertificacao

MÉTODO getDataUTC

Esta função retorna a data UTC (Coordinated Universal Time) da assinatura digital referente ao índice passado como parâmetro.

A data UTC é a data no meridiano de Greenwich que é a referência do tempo zero onde todos os países se baseiam para calcular os seus fuso-horários.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

BSTR getDataUTC (int __index);

Parâmetros:

__index	int	Índice da data UTC da assinatura iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** Data UTC da assinatura **Erro:** vazio

MÉTODO getDataCalculada

Esta função retorna a data calculada sobre a data UTC da assinatura digital referente ao índice passado como parâmetro. O cálculo é feito utilizando-se as configurações do usuário windows que está utilizando o computador, para o

cálculo são obtidas informações como por exemplo o fuso-horário configurado pelo usuário.

Para ter acesso a essa função é necessária antes a verificação da assinatura pelos métodos ñverifiqueAssinaturaò ou ñverifiqueAssinaturaMemò

Assinatura do método:

BSTR getDataCalculada(int __index);

Parâmetros:

__index	Int	Índice da data Calculada da assinatura iniciando da posição 0.

Retorno:

retorno	BSTR	Data Calculada com configurações do usuário windows da assinatura

MÉTODO: setFormatoDadosMemoria

Seta o formato de codificação/decodificação que será utilizado na classe Verificador. Este formato será utilizado na hora de decodificar o conteúdo assinado em memória que será verificado.

Se esta função nunca for utilizada, o valor default utilizado será a codificação/decodificação em HEXADECIMAL.

Quando se deseja setar qual o formato de codificação/decodificação que será utilizado, deve-se chamar este método antes dos métodos que realizam a verificação da assinatura em memória.

Assinatura do método:

int setFormatoDadosMemoria(int __formato);

Parâmetros:

__formato	int	Formato a ser utilizado pela classe. Os valores podem ser: 0 = HEXADECIMAL (default) 1 = BASE64

Retorno:

retorno	int	**Sucesso:** 1 **Erro:** ERRO_FORMATO_INVALIDO

MÉTODO getHashDocumentoOriginal

Retorna o hash do conteúdo original assinado que está presente no campo content do content info da assinatura.

Assinatura do método:

BSTR getHashDocumentoOriginal (int __index);

Parâmetros:

__index	Int	Índice da assinatura que se deseja pegar o hash do conteúdo original presente iniciando da posição 0.

Retorno:

retorno	BSTR	**Sucesso:** hash SHA-1 em hexadecimal **Erro:** vazio

MÉTODO verifiqueAssinaturaDetached

Este método verifica a assinatura digital de um arquivo assinado comparando os dados do arquivo original com o arquivo assinado. É necessário o documento original para extrair o hash, pois a assinatura é detached, ou seja não contem o dado assinado.

Para cada verificação é gerado um objeto contendo: certificado digital utilizado na assinatura, arquivo original, descrição, ***status*** de verificação e status do certificado.

Se todos os dados forem compatíveis esta função retornará SUCESSO, no entanto isto não significa que a assinatura, ou certificados digital são válidos.

Assinatura do método:

verifiqueAssinaturaDetached(BSTR __arqAssinado, BSTR __arqOriginal, int *__status)

Parâmetros:

arqAssinado	BSTR	Caminho do documento assinado.
arqOriginal	BSTR	Caminho do documento original.
status	int	

Retorno:

status	int	**Sucesso:** 1 **Erros comuns:** Erros: números negativos Verificar lista de constantes de erro.

MÉTODO verifiqueAssinaturaMemDetached

Este método verifica a assinatura digital de arquivo assinado em um bloco de memória comparando o hash do documento original com o hash da assinatura. É necessário o documento original para extrair o hash, pois a assinatura é detached, ou seja não contem o dado assinado.

Para cada verificação é gerado um objeto contendo: certificado digital utilizado na assinatura, descrição, ***status*** de verificação e status do certificado.

Se todos os dados forem compatíveis esta função retornará SUCESSO, no entanto isto não significa que a assinatura, ou certificados digital são válidos.

Assinatura do método:

verifiqueAssinaturaMemDetached(BSTR __memAssinada, BSTR __memOriginal, int *__status)

Parâmetros:

memAssinada	BSTR	Caminho do bloco de memória onde foi a
memOriginal	BSTR	
status	int	

Retorno:

status	int	**Sucesso:** 1 **Erros comuns:**números negativos *verificar valores de retorno dos erros na tabela de erros

MÉTODO isArquivoAssinadoDetached

Este método verifica se o arquivo assinado digitalmente está no formato detached, ou seja, não possui o conteúdo assinado em sua estrutura.

Assinatura do método:

isArquivoAssinadoDetached (BSTR __arquivo)

Parâmetros:

__arquivo	BSTR	Caminho do arquivo de assinatura.

Retorno:

status	int	**Sucesso:** 1 caso detached 0 caso atached **Erros comuns:**números negativos *verificar valores de retorno dos erros na tabela de erros

MÉTODO getLCR

Retorna um objeto CRL que foi encontrado na estrutura do documento assinado durante a verificação. Essa função somente deve ser utilizada após a verificação de um arquivo assinado.

Assinatura do método:

ICRL getLCR (int __posicao)

Parâmetros:

__index	Int	Índice da CRL que se deseja iniciando da posição 0.

Retorno:

Retorno	ICRL	**Sucesso:** Objeto do tipo ICRL **Erro:**vazio

MÉTODO getCountCRLs

Retorna a quantidade de CRLs internas a estrutura do arquivo assinado. Arquivos assinados podem conter em sua estrutura CRLs.

Assinatura do método:

int getCountCRLs ()

Parâmetros: Não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número de CRLs presentes no arquivo **Erro: 0**

MÉTODO getCountCertificadosCadeia

Retorna a quantidade de certificados internos da estrutura do documento assinado. Documentos assinados podem conter em sua estrutura certificados adicionais além do certificado do signatário. Usualmente certificados da cadeia são adicionas a estrutura do arquivo assinado para verificação a longo prazo. Este método somente deve ser executado após o procedimento de verificação de um documento assinado.

Assinatura do método:

int getCountCertificadosCadeia ()

Parâmetros: Não possui parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número de Certificados presentes no arquivo **Erro: Consultar tabela de erros em anexo.**

MÉTODO getCertificadoCadeia

Retorna um objeto Certificado que foi encontrado na estrutura do documento assinado durante a verificação. Essa função somente deve ser utilizada após a verificação de um arquivo assinado. Pode ser utilizado em um laço com o método ñ**getCountCertificadosCadeia**ò

Assinatura do método:

ICertificado getCertificadoCadeia (int __posicao)

Parâmetros:

__index	Int	Índice do Certificado que se deseja iniciando da posição 0.

Retorno:

Retorno	ICertificado	**Sucesso:** Objeto do tipo ICertificado **Erro:**Objeto Nulo

MÉTODO setInstalarCadeia

Seta flag que será utilizada na verificação. Está flag indica se a cadeia de certificação presente na assinatura no momento da verificação deve ou não ser instalada no repositório do Windows. Se desejar instalar a cadeia, deve-se chamar este método antes dos métodos que realizam a verificação da assinatura.

Assinatura do método:

void setInstalarCadeia (bool __instalarCadeia)

Parâmetros:

__instalarCadeia	BOOL	Indica se deve ou não instalar a cadeia de dentro da assinatura no repositório do Windows.

Retorno:

Método sem retorno

MÉTODO setBaixarLCR

Seta flag que será utilizada na verificação. Está flag indica se a(s) LCR(s) não presentes na assinatura ou no repositório do windows no momento da verificação deve(m) ou não ser(em) baixada(s) e instalada(s) no repositório do Windows. Se desejar baixar e instalar a(s) LCR(s) deve-se chamar este método antes dos métodos que realizam a verificação da assinatura. Baixa LCR somente se não encontrar LCR ñválidaòdentro da assinatura ou no repositório.

Assinatura do método:

void setBaixarLCR (bool __baixarLCR)

Parâmetros:

__baixarLCR	BOOL	Indica se deve ou não baixar e instalar a(s) LCR(s) no repositório do Windows.

Retorno:

Método sem retorno

MÉTODO setInstalarCertAssinador

Seta flag que será utilizada na verificação. Está flag indica se o certificado do assinante deve ou não ser instalado no repositório ñOther Peopleòdo Windows. Se desejar instalar o certificado do assinante, deve-se chamar este método antes dos métodos que realizam a verificação da assinatura. Caso o arquivo assinado contenha mais de uma assinatura, irá instalar o certificado assinante de cada uma delas.

Assinatura do método:

void setInstalarCertAssinador (bool __instalarCertAssinador)

Parâmetros:

__instalarCertAssinador	BOOL	Indica se deve ou não instalar o certificado do assinante no repositório "Other People".

Retorno:

Método sem retorno

MÉTODO: setLoginProxy

Seta qual deverá ser o login do usuário que deverá ser utilizado pela função baixarCRL no caso de utilização de proxy.

Assinatura do método:

void setLoginProxy(BSTR __login);

Parâmetros:

__login	BSTR	é o login que deverá ser utilizado pelo proxy na função baixarCRL

MÉTODO: setSenhaProxy

Seta qual deverá ser a senha do usuário que deverá ser utilizado pela função baixarCRL no caso de utilização de proxy.

Assinatura do método:

void setSenhaProxy(BSTR __senha);

Parâmetros:

__senha	BSTR	é a senha que deverá ser utilizada pelo proxy na função baixarCRL

4.5 Classe Carimbo (ClassID: 83C5AF6B-0CEF-4A58-9ACC-009ABEF94260)

A classe Carimbo exporta os métodos do carimbo de tempo referente a uma assinatura digital. Quando um arquivo assinado é verificado e o mesmo possui um carimbo de tempo, o objeto Carimbo fica disponível através do método getCarimboAssinatura. O objeto retornado é da classe Carimbo, possibilitando a extração de todos os dados do carimbo de tempo.

Os métodos da classe Carimbo estão descritos abaixo:

MÉTODO getNumeroSerial

Esta função tem o objetivo de recuperar o número serial do carimbo.
O número de serie representa a ordem em que o carimbo foi emitido, está ordem é crescente. Para saber se existe um número de série realize uma chamada para a função isOrdering(), se a mesma retornar verdadeiro existe um número de série.

Assinatura do método:

BSTR getNumeroSerial()

Parâmetros:

Não possui.

Retorno:

numeroSerie	BSTR	**Sucesso:** Número de serie que representa ordem que o carimbo foi emitido.

MÉTODO getVersao

Esta função tem o objetivo de recuperar a versão do carimbo de tempo.

Assinatura do método:

int getVersao ()

Parâmetros:

Não possui.

Retorno:

versao	int	**Sucesso:** Número que representa a versão do carimbo de tempo.

MÉTODO getAccuracyMicrosegundos

Esta função tem o objetivo de recuperar a precisão em microssegundos que o relógio da carimbadora tinha quando emitiu determinado carimbo. Este é um campo opcional do Carimbo de Tempo.

Assinatura do método:

BSTR getAccuracyMicrosegundos ()

Parâmetros:

Não possui.

Retorno:

microsegundos	BSTR	**Sucesso:** String contendo o valor microsegundos do carimbo.

MÉTODO getAccuracyMiliSegundos

Esta função tem o objetivo de recuperar a precisão em milisegundos que o relógio da carimbadora tinha quando emitiu determinado carimbo. Este é um campo opcional do Carimbo de Tempo.

Assinatura do método:

BSTR getAccuracyMicrosegundos ()

Parâmetros:

Não possui.

Retorno:

milisegundos	BSTR	**Sucesso:** String contendo o valor milisegundos do carimbo.

MÉTODO getAccuracySegundos

Esta função tem o objetivo de recuperar a precisão em segundos que o relógio da carimbadora tinha quando emitiu determinado carimbo. Este é um campo opcional do Carimbo de Tempo.

Assinatura do método:

BSTR getAccuracySegundos()

Parâmetros:

Não possui.

Retorno:

segundos	BSTR	**Sucesso:** String contendo o valor em segundos do campo accuracy do carimbo.

MÉTODO getCertificadoCarimbadora

Esta função tem o objetivo de retornar o certificado utilizado pela carimbadora. Retornando este objeto tem-se acesso a todos os métodos da classe Certificado.

Assinatura do método:

ICertificado getCertificadoCarimbadora ()

Parâmetros:

Não possui.

Retorno:

certificado	ICertificado	**Sucesso:** Certificado da carimbadora. **Erro:** NULL.

MÉTODO getDataCalculada

Esta função tem o objetivo de retornar a data do carimbo de tempo em relação ao fuso horário local.

Assinatura do método:

BSTR getDataCalculada()

Parâmetros:

Não possui.

Retorno:

datacalculada	BSTR	**Sucesso:** String com a data calculada. **Erro:** "".

MÉTODO getDataUTC

Esta função tem o objetivo de retornar a data do carimbo de tempo no formato UTC, que é a data utilizada pela carimbadora por padrão.

Assinatura do método:

BSTR getDataUTC()

Parâmetros:

Não possui.

Retorno:

datautc	BSTR	**Sucesso:** String com a data calculada. **Erro:** ””.

MÉTODO getNonce

Esta função tem o objetivo de retornar o nonce do carimbo de tempo. Nonce é um número aleatório gerado no momento do carimbo de tempo.

Assinatura do método:

int getNonce()

Parâmetros:

Não possui.

Retorno:

nonce	int	**Sucesso:** int representando o nonce. **Erro:** ””.

MÉTODO getPolitica

Esta função tem o objetivo de recuperar o OID da política de datação utilizada pela carimbadora.

Assinatura do método:

BSTR getPolitica()

Parâmetros:

Não possui.

Retorno:

politica	BSTR	**Sucesso:** conterá o OID da política. **Erro:** ””.

MÉTODO getStatus

Esta função tem o objetivo retornar o estado do carimbo de tempo, se o mesmo está válido ou não.

Assinatura do método:

int getStatus()

Parâmetros:

Não possui.

Retorno:

status	BSTR	**Sucesso:** 1. **Erro:** Constantes de erro. Erros mais comuns: OBJETO_NAO_INSTANCIADO. ERRO_INTERNO_CRYPTO ERRO_ASSINATURA ERRO_ASSINATURA_TST_INVALIDA HASH_DIFERENTES ERRO_VERIFICANDO_TIME_STAMP

MÉTODO getTSAHash

Esta função tem o objetivo de recuperar o hash da assinatura que está dentro do carimbo de tempo.

Assinatura do método:

BSTR getTSAHash()

Parâmetros:

Não possui.

Retorno:

hash	BSTR	**Sucesso:** string contendo o hash da assinatura. **Erro:** "".

MÉTODO isOrdering

Esta função tem o objetivo de recuperar o campo *ordering* do recibo. Este campo indica se existe uma seqüência de número serial do recibo.

Assinatura do método:

boolean isOrdering()

Parâmetros:

Não possui.

Retorno:

boolean	BOOL	**O retorno pode ser:** 0 – falso 1 - verdadeiro

4.6 Classe CRL (ClassID: 86DE8D4F-D663-4395-82E9-051D79766FB9)

A classe CRL exporta métodos de acesso aos dados de um arquivo CRL (Lista certificados revogados) referente a um certificado digital. A revogação é um dispositivo no qual um certificado pode perder a validade em um momento anterior a sua expiração, devido a algum problema com o mesmo, exemplo: o dono perder o smart card que contem seu certificado. No caso após comunicar a sua Autoridade Certificadora a mesma se encarrega de emitir uma lista de certificados revogados com os certificados que foram revogados.

Os métodos da classe CRL estão descritos abaixo:

MÉTODO inicializar

Esta função tem o objetivo criar um objeto CRL através do caminho de um arquivo CRL em disco.

Assinatura do método:

int inicializar(BSTR __arquivo)

Parâmetros:

__arquivo	BSTR	Caminho em disco para o local onde se encontra o arquivo CRL.

Retorno:

retorno	int	**Sucesso:** 1 Caso a inicialização tenha acontecido corretamente. **Erro:** Números negativos conforme tabela abaixo.

MÉTODO getDataInicio

Retorna a data em que a CRL foi emitida.

Assinatura do método:

BSTR getDataInicio()

Parâmetros: Não possui parâmetros

Retorno:

retorno	BSTR	**Sucesso:** Data de emissão da LCR no formato dd/mm/aaaa. **Erro:** vazio.

MÉTODO getDataTermino

Retorna a data em que a CRL vai expirar.

Assinatura do método:

BSTR getDataTermino()

Parâmetros: Não possui parâmetros

Retorno:

retorno	BSTR	**Sucesso:** Data de expiração da LCR no formato dd/mm/aaaa. **Erro:** vazio.

MÉTODO getAlgoritmoAssinatura

Retorna a data em que a CRL vai expirar.

Assinatura do método:

BSTR getAlgoritmoAssinatura()

Parâmetros: Não possui parâmetros

Retorno:

retorno	BSTR	**Sucesso:** Nome do algoritmo de assinatura utilizado na assinatura da LCR. **Erro:** vazio.

MÉTODO: getEmissor

Retorna o valor do Emissor (CN,O,OU,L,S,C,E) da LCR que o objeto representa

Assinatura do método:

BSTR getEmissor();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o Emissor da LCR **Falha:** retorna vazio

MÉTODO: getEmissorCN

Retorna o valor do CN (Nome Comum) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorCN();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o CN do Emissor da LCR **Falha:** retorna vazio

MÉTODO: getEmissorO

Retorna o valor do O (Organização) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorO();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o O do Emissor da LCR **Falha:** retorna vazio

MÉTODO: getEmissorOU

Retorna o valor do OU (Unidade da Organização) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorOU();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o OU do Emissor do LCR **Falha:** retorna vazio

MÉTODO: getEmissorL

Retorna o valor do L (Localidade) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorL();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o L do Emissor do LCR **Falha:** retorna vazio

MÉTODO: getEmissorS

Retorna o valor do S (Estado) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorS();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o S do Emissor do LCR **Falha:** retorna vazio

MÉTODO: getEmissorC

Retorna o valor do C (País) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorC();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o C do Emissor da LCR **Falha:** retorna vazio

MÉTODO: getEmissorE

Retorna o valor do E (e-mail) do Emissor da LCR que o objeto representa

Assinatura do método:

BSTR getEmissorE();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	BSTR	**Sucesso:** Retorna o E do Emissor do LCR **Falha:** retorna vazio

MÉTODO: getNumeroLista

Retorna o número de emissão da LCR

Assinatura do método:

int getNumeroLista ();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número representante da LCR **Falha:** Números negativos conforme tabela abaixo.

MÉTODO: getVersao

Retorna a versão da LCR

Assinatura do método:

int getVersao();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Sucesso:** Número representante da versão da LCR **Falha:** Números negativos conforme tabela abaixo.

MÉTODO: visualizarLCR

Utilizando esta função uma interface gráfica é apresentada ao cliente com os dados da LCR.

Assinatura do método:

int visualizarLCR();

Parâmetros: Esta função não tem parâmetros.

Retorno:

Retorno	int	**Sucesso:** 1 em caso de sucesso **Falha:** Números negativos conforme tabela abaixo.

Valores de Retorno e seus significados:

-30001	ARQUIVO_INEXISTENTE ï Arquivo não existe
-30002	ARQUIVO_INVALIDO ï Arquivo Inválido (formato inválido)
-30003	ARQUIVO_VAZIO ï Arquivo está vazio
-30004	EXTENSAO_INVALIDA ï Extensão inválida
-30005	CERTICADO_NAO_ENCONTRADO ï Certificado digital não foi encontrado
-30006	CERTIFICADO_SEM_CHAVE_PRIVADA ï Certificado utilizado não possui chave privada
-30007	ERRO_INTERNO_CRYPTO ï Erro interno da biblioteca CryptoAPI.
-30008	ARQUIVO_JA_EXISTE ï Arquivo já existe
-30010	NOME_ARQUIVO_NAO_ENCONTRADO ï Nome do arquivo original não encontrado na assinatura
-30011	HASH_DIFERENTES ï Hashs não conferem
-30012	CERTIFICADO_INVALIDO ï Certificado inválido
-30013	CONTEUDO_NAO_VERIFICADO ï Conteúdo da assinatura não foi verificado
-30014	CERTIFICADO_INVALIDO_ASSINATURA ï Certificado não pode assinar documentos
-30015	ASSINATURA_SEM_NOME_ARQUIVO ï Assinatura não contém nome do arquivo
-30016	DESTINO_INVALIDO ï Caminho do arquivo inválido.
-30017	HASH_NAO_ENCONTRADO ï Certificado não foi encontrado
-30018	STATUS_INEXISTENTE ï Status de verificação não existe
-30019	ERRO_SEM_ARQUIVO_ASSOCIADO ï Não existe programa associado a extensão
-30020	BLOCO_MEMORIA_INVALIDO ï Bloco de Memória assinada inválido
-30021	DESCRICAO_ARQUIVO_NAO_ENCONTRADO ï não foi possível extrair a descrição do arquivo
-30022	DATA_ARQUIVO_NAO_ENCONTRADO ï não foi possível extrair a data do arquivo
-30023	ERRO_CODIFICANDO_ATRIBUTO_AUTENTICADO ï erro inserindo atributo autenticado na assinatura
-30024	ERRO_CONVERSAO ï erro na conversão de tipos string para BSTR
-30025	ERRO_CONTEUDO_VAZIO ï conteúdo a ser assinado vazio
-30026	ERRO_DATA_INVALIDA ï data fornecida não existe
-30027	ERRO_FORMATO_INVALIDO ï formato de codificação/decodificação inválido
-30028	ERRO_CALCULANDO_HASH ï erro calculando o hash dos dados
-30029	ERRO_ASSINATURA ï assinatura inválida.
-30030	ERRO_SENHA_VAZIA - senha do certificado vazia
-30031	OBJETO_NAO_INSTANCIADO ï objeto com não foi inicializado

-30032	ERRO_CARREGANDO_KEY_CONTAINER ï Não foi possível encontrar o container da chave privada
-30033	ERRO_ACESSANDO_CHAVE_PRIVADA ï Não foi possível a chave privada.
-30034	ERRO_ASSINATURA_TST_INVALIDA ï A assinatura do carimbo de tempo não é integra.
-30035	ERRO_ESTRUTURA_TSTINFO ï A estrutura do carimbo de tempo está corrompida. Não foi possível decodificar o mesmo.
-30036	NAO_EXISTE_TIME_STAMP ï A assinatura não possui carimbo de tempo.
-30037	ERRO_VERIFICANDO_TIME_STAMP ï Ocorreu um erro interno ao verificar a assinatura do carimbo de tempo.
-30038	ERRO_DECODIFICANDO_DATA_TIME_STAMP ï Não foi possível decodificar a data do carimbo de tempo.
-30039	ERRO_ADQUIRINDO_PROVEDOR ï Erro ao adquirir o provedor do certificado.
-30040	ERRO_CONVERTENDO_ORIGINAL_HEXA_BIN ï Erro ao converter o arquivo original em uma verificação de arquivo detached de hexadecimal para binário.
-30041	ERRO_CONVERTENDO_ORIGINAL_BASE64 - Erro ao converter o arquivo original em uma verificação de arquivo detached de base64 para binário.
-30042	ERRO_ARQUIVO_ORIGINAL_VAZIO ï Arquivo original não possui conteúdo.
-30043	ERRO_ARQUIVO_ATACHED ï O arquivo de assinatura é atached.
-201	ERRO_PONTEIRO_STORE_NULO - Ponteiro do store em questao nulo
-202	ERRO_PONTEIRO_CERTIFICADO_NULO - Ponteiro do certificado em questão nulo
-203	ERRO_OBJ_STORE_NULO - O objeto store está vazio
-204	ERRO_LISTA_CERTIFICADOS_VAZIA - Lista de Certificados vazia
-301	ERRO_WSASTARTUP_FAILED - Falha no Socket
-302	ERRO_WSANOTINITIALISED - Socket ainda não pronto
-303	ERRO_WSAENETDOWN - Rede fora do ar
-304	ERRO_WSAHOST_NOT_FOUND - Host não encontrado
-305	ERRO_WSATRY_AGAIN - Tente novamente
-306	ERRO_WSANO_RECOVERY - Erro não recuperado
-307	ERRO_WSANO_DATA - Nenhum dado do tipo requerido
-308	ERRO_WSAEINPROGRESS - operação está em progresso
-309	ERRO_WSAEAFNOSUPPORT - endereço não suportado pelo protocolo
-310	ERRO_WSAEFAULT - Endereço errado
-311	ERRO_WSAEINTR - Chamada da função interrompida
-401	ERRO_ABRIR_ARQUIVO - Não foi possível abrir ou verificar o tamanho do arquivo
-402	ERRO_FORMATO_ARQUIVO_BRY - Formato do arquivo BRy invalido
-501	ERRO_POSICAO_CERT_INEXISTENTE - Não tem certificados nesta posição da lista
-502	ERRO_MOSTRAR_CERTIFICADO - Não foi possível mostrar o certificado

-601	ERRO_CARREGANDO_CRL - Não foi possível carregar a CRL
-602	ERRO_CARREGANDO_FINALIDADE - Não foi possível carregar o uso avançado da chave
-603	ERRO_CARREGANDO_USO_CHAVE - Não foi possível carregar o uso da chave
-604	ERRO_CARREGANDO_CONTEXTO_CERTIFICADO - Não foi possível carregar o contexto do certificado
-605	ERRO_CARREGANDO_VALIDADE - Não foi possível carregar a validade
-606	ERRO_CARREGANDO_DADOS - Não foi possível carregar os dados do certificado
-607	ERRO_CARREGANDO_PONTEIRO_CERTIFICADO - Não foi possível carregar ponteiro do certificado
-608	ERRO_CARREGANDO_CERTIFICADO_EMISSOR - O certificado emissor não foi carregado
-609	ERRO_ALGUNS_VALORES_NAO_CARREGADOS - Alguns valores não foram carregados
-610	ERRO_CARREGANDO_EXTENSAO_ICPBRASIL ï Erro carregando os valores de CPF e RG das extensões nome alternativo do assunto de certificados ICP-BRASIL
-611	ERRO_CARREGANDO_EXTENSAO_POLICY ï Erro carregando os valores da extensão Diretivas dos certificados (Certificate Policies)
-612	ERRO_CARREGANDO_USO_EXTENDIDO_CHAVE ï Ocorrem problemas ao decodificar o uso estendido da chave.
-613	ERRO_DECODIFICANDO_BASIC_CONSTRAINTS ï Ocorreram problemas ao decodificar a extensão basic contraints.
-701	ERRO_IDENTIFICACAO_URL - Não conseguiu identificar a url
-702	ERRO_INSTALANDO_CRL - Não foi possível instalar a CRL
-703	ERRO_CRL_AINDA_NAO_VALIDA ï A data atual é menor que a da publicacao da crl
-704	ERRO_CRL_EXPIRADA ï A data atual é maior que a data de validade da CRL
-705	ERRO_CRL_NAO_INSTALADA ï A crl não está instalada no computador
-706	ERRO_CAMINHO_CRL - Problemas com o caminho da CRL
-707	ERRO_LENDO_ARQUIVO_CRL - Não conseguiu ler o arquivo da CRL
-708	ERRO_ABRINDO_ARQUIVO_CRL - Não conseguiu abrir o arquivo da crl
-709	ERRO_CARREGANDO_CADEIA_CERTIFICACAO - não conseguiu carregar a cadeia de certificação
-710	ERRO_CRL_NAO_VALIDA_PARA_CERTIFICADO ï A CRL utilizada não é válida para verificar o certificado digital.
-801	ERRO_HEX_BIN - Erro tranformando de hexadecimal para binario
-802	ERRO_BIN_HEX - Erro tranformando de binario para hexadecimal
-803	ERRO_DECODIFICANDO_BASE_64 ï Erro transformando de base 64 para binário
-804	ERRO_BASE64_BIN ï Erro decodificando de Base64 para binário.
-901	ERRO_CERTIFICADO_REVOGADO ï O certificado esta revogado
-902	ERRO_CERTIFICADO_AINDA_NAO_VALIDO - Data atual é menor que a da publicacao do certificado
-903	ERRO_CERTIFICADO_EXPIRADO - Data atual é maior do que o prazo de validade do certificado

-1001	ERRO_ADICIONAR_CERT_STORE - Não foi possível adicionar certificado ao Store
-1002	ERRO_ADICIONAR_CERT_LISTA_CERTIFICADOS - Erro ao adicionar um certificado na lista de certificados.
-1101	ERRO_PARAMETRO_TIPO_INVALIDO - Valor inválido para o parâmetro tipo
-1102	ERRO_FLAG_NAO_SUPORTADA - Flag não suportada pelo método.
-1201	ERRO_ACESSO_NEGADO ï Acesso a crl negado
-1202	ERRO_AUTENTICACAO_NO_PROXY ï Erro de autenticação no proxy
-1203	ERRO_NO_SERVIDOR ï erro no servidor que hospeda a crl
-1204	ERRO_ACESSO_PROIBIDO ï acesso proibido ao endereço da crl
-1205	ERRO_PAGINA_NAO_ENCONTRADA ï endereço da crl não encontrado
-1206	ERRO_CONECTANDO_URL ï erro conectando a URL da CRL
-1207	ERRO_TIMEOUT ï Timeout na comunicação com servidor da CRL
-1208	ERRO_URL_INVALIDA ï URL da CRL inválida
-1209	ERRO_CONSULTA_INVALIDA ï consulta http inválida
-1210	ERRO_RESPOSTA_SERVIDOR ï resposta do servidor inválida
-1211	ERRO_SERVIDOR_REQUER_CERTIFICADO_CLIENTE ï servidor requer autenticação do cliente com certificado digital
-1212	ERRO_CONEXAO_TERMINADA ï a conexão com a internet foi encerrada
-1213	ERRO_CONEXAO_RESETADA ï a conexão com a internet foi resetada
-1214	ERRO_INTERNO_INTERNET ï erro interno nas bibliotecas de acesso a internet
-1215	ERRO_DECODIFICANDO_ASSUNTO ï Ocorreram problemas ao decodificar o assunto
-1216	ERRO_DECODIFICANDO_EMISSOR ï Ocorreram problemas ao decodificar o emissor

Apêndice A

Este documento tem por objetivo definir o formato do arquivo que conterá os certificados a serem instalados através da função de instalação de certificados, adicionarDoArquivo, pertencente a classe *repositorio*. A especificação desta função está detalhada na descrição da Classe *repositorio*.

Considerações:

1. O arquivo pode conter quantos certificados for necessário.
2. Deverá ser especificado o tipo do certificado a ser instalado: se é correspondente a uma AC raiz, se é uma AC intermediária ou se é apenas um certificado normal.
3. Não é necessária a identificação de cada certificado para uma possível busca, pois se considera que cada tentativa de instalação irá tentar instalar todos os certificados existentes neste arquivo.

Formato:

O arquivo possuirá várias entradas. Cada entrada deverá estar numa linha do arquivo e cada uma corresponderá a um certificado.

Cada linha possui o seguinte formato:

<<Tipo do Certificado><Nome do Certificado><Certificado Base64 e em Hexa>>

Valores aceitos em Tipo do Certificado: Root, ca, normal.

Apêndice B

Dicas:

__ : significa parâmetro
_ : variável local
BSTR: é uma string unicode

Para utilizar em Builder e em delphi a DLL, há a necessidade de registrá-la através do comando regsvr32 ñbrysignerCOM.dllò

Para funcionar em delphi e builder você deve importar uma classe, através do seguinte comando: Project→ import→ typelibrary → <selecionar a DLL que aparece> → clicar em create unit.

Acrescentar em uses:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, SIGNERCOMSDKLib_TLB;
```

é

```
  public
    signer : IAssinador;
    verifier  : IVerificador;
    certificado : ICertificado;
    repositorio : IRepositorio;
    carimbo: ICarimbo;
  end;
```