Manual do Desenvolvedor

Criptografia de Arquivos do WebTA

Versão 1.5

Índice

1. Objetivo 3
2. Processo 3
3. API de Criptografia - Biblioteca Java 4
   3.1 WEBTACryptoUtil 4
      3.1.1 **decodeKeyFile** 4
      3.1.2 **getIdFromDecodedFileKey** 5
      3.1.3 **getKeyFromDecodedFileKey** 5
   3.2 WEBTAOutputStream 6
      3.2.1 **WEBTAOutputStream** 6
      3.2.2 **write** 6
      3.2.3 **size** 7
      3.2.4 **flush** 7
      3.2.5 **close** 7
   3.3 WEBTAInputStream 8
      3.3.1 **WEBTAInputStream** 8
      3.3.2 **read** 8
      3.3.3 **close** 9
   3.4 CryptoException, ParameterException e GZipException 10
4. API de Criptografia - Biblioteca C - Windows 11
   4.1 Decodificação dos arquivos de chaves 11
      4.1.1 **fDecodeKeyFile** 11
      4.1.2 **fDecodeKeyFileEx** 12
   4.2 Criptografia de arquivos 13
      4.2.1 **fInitEncoder** 13
      4.2.2 **fWritedata** 13
      4.2.3 **fCloseEncoder** 13
   4.3 Descriptografia de arquivos 15
      4.3.1 **fInitDecoder** 15
      4.3.2 **fReadData** 15
      4.3.3 **fCloseDecoder** 15
   4.4 Utilização em uma classe .NET 17

Manual para utilização da Criptografia de Arquivos do WebTA

1. Objetivo

Este documento tem como objetivo descrever os procedimentos para ativação/utilização do mecanismo de criptografia de arquivos utilizados na Transferência de Arquivos – WebTA.

A Criptografia do WebTA tem como principal objetivo aumentar a segurança da informação dos dados que são trafegados entre a Empresa e o Banco.

Por exemplo, quando o Sistema da Empresa cria um arquivo de remessa, gera-o em formato texto. Este formato permite a visualização de seu conteúdo, bem como a edição do mesmo. Com as novas bibliotecas de Criptografia é possível customizar o Sistema da Empresa para que gere os arquivos de remessa com conteúdo criptografado.

Neste manual existem seções para os desenvolvedores da aplicação com as descrições das interfaces de programação (Linguagens C e Java) das funções/métodos para a criação e leitura dos arquivos criptografados que serão intercambiados entre o Sistema da Empresa e o WebTA.

2. Processo

Abaixo são descritos os passos necessários para a utilização da Criptografia de arquivos do WebTA:

1º. Entre em contato com a Central de Atendimento e peça a disponibilização deste serviço;

2º. O Master da Empresa deverá acessar o Ambiente Gerencial do WebTA e gerar as chaves de criptografia. Neste processo o Sistema irá gerar 2 arquivos de chaves:

- TRANSFERENCIAaaaammddhhmm.bin: contém a chave e o ID necessários para realizar a transmissão/recepção automática de arquivos.

- CRIPTOGRAFIAaaaammddhhmm.bin: contém a chave utilizada para criptografar os arquivos de remessa ou descriptografar o conteúdo dos arquivos de retorno.

Estes arquivos serão gerados criptografados com a senha que é solicitada ao Master. Esta senha precisará ser passada à API (*Application Programming Interface)* de criptografia, para que a mesma possa acessar os arquivos das chaves.

Onde “aaaammddhhmm” representam a data e hora de geração dos arquivos.

3º. Peça ao Desenvolvedor de seu Sistema para que utilizando este Kit de Desenvolvimento e o arquivo de chave “CRIPTOGRAFIAaaaammddhhmm.bin”, faça os ajustes necessários no seu Sistema para que o mesmo gere e leia os arquivos no modo criptografado;

4º. O Master da Empresa deverá acessar o Ambiente Gerencial do WebTA e ativar a utilização da transmissão dos arquivos no modo criptografado.

3. API de Criptografia - Biblioteca Java

A biblioteca Java de Criptografia de Arquivos do WebTA disponibiliza para o Desenvolvedor as classes:

- WEBTACryptoUtil: classe para decodificar os arquivos de chaves do cliente;
- WEBTAOutputStream: classe para gerar os arquivos de remessa comprimidos e criptografados no formato WebTA;
- WEBTAInputStream: classe para descriptografar e descomprimir os arquivos de retorno criptografados do WebTA;
- CryptoException, GZipException e ParameterException: classes para tratamento de exceções das anteriormente descritas.

No *Kit* de Desenvolvimento são distribuídos o arquivo "webta-cripto-v.1.4.jar", que é a biblioteca propriamente dita, e o Javadoc correspondente.

É pré-requisito para sua utilização o Java SDK 1.3.1 ou superior.

3.1 WEBTACryptoUtil

Esta classe contém os métodos necessários para decodificar os arquivos de chaves que foram gerados pelo Master da Empresa no Ambiente Gerencial do site WebTA.

***Pacote**:* `br.com.bradesco.webta.security.crypto.WEBTACryptoUtil`

Descrição dos métodos:

3.1.1 **decodeKeyFile**

Método estático que decodifica os arquivos gerados no Ambiente Gerencial do WebTA: CRIPTOGRAFIAaaaammddhhMM.bin e TRANSFERENCIAaaaammddhhMM.bin.

```
public static byte[] decodeKeyFile(File file, String senha) throws IOException,
CryptoException
```

Parâmetros de entrada:

- file: descritor do arquivo que contém a chave de criptografia;
- senha: senha utilizada para abrir o arquivo.

Retorno:

Array de bytes contendo o conteúdo do arquivo passado como parâmetro. Se for o arquivo CRIPTOGRAFIAaaaammddhhMM.bin o array conterá a chave que será utilizada como parâmetro nos métodos das classes de Criptografia e Descriptografia. Se for o arquivo Transferenciaaaaammddhhmm.bin deve-se utilizar os métodos descritos a seguir para obter o Id e a chave de criptografia necessárias para o processo de autenticação da transferência automática de arquivos.

Exceções:

- IOException: se ocorrer um erro de I/O;
- CryptoException: erro ao decodificar o arquivo.

3.1.2 **getIdFromDecodedFileKey**

Método estático que obtém o Id do Cliente a partir do conteúdo decodificado do arquivo TRANSFERENCIAaaaammddhhMM.bin. Este Id é necessário no processo de Autenticação da Transferência Automática de Arquivos.

```
public static String getIdFromDecodedFileKey(byte[] key) throws
ParameterException
```

Parâmetro de entrada:

- key: Chave de Transferência decodificada pelo método decodeKeyFile, a partir do arquivo TRANSFERENCIAaaaammddhhMM.bin.

Retorno:

String contendo o Id do Cliente.

Exceções:

- ParameterException: se a chave for inválida.

3.1.3 **getKeyFromDecodedFileKey**

Método estático que obtém a Chave de Criptografia a partir do conteúdo decodificado do arquivo TRANSFERENCIAaaaammddhhMM.bin. Esta chave é necessária no processo de Autenticação da Transferência Automática de Arquivos.

```
public static byte[] getKeyFromDecodedFileKey(byte[] key) throws
ParameterException
```

Parâmetro de entrada:

- key: Chave de Transferência decodificada pelo método decodeKeyFile, a partir do arquivo TRANSFERENCIAaaaammddhhMM.bin.

Retorno:

Array de bytes contendo a Chave de Criptografia.

Exceções:

- ParameterException: se a chave for inválida.

Exemplo de utilização da classe:

```
...
//Decodifica arquivo que contém a chave
//necessária para criptografar os arquivos de
//remessa e descriptografar os arquivos de
//retorno
byte[] fileKey = WEBTACryptoUtil.decodeKeyFile(new File
("e:/CRIPTOGRAFIA200805191537.bin"), "a2owdf089");

...

//Decodifica arquivo que contém parâmetros
//necessários para o processo de Autenticação
//na Transferência Automática de Arquivos
byte[] fileKey = WEBTACryptoUtil.decodeKeyFile(new File
("e:/TRANSFERENCIA200805191537.bin"), "a2owdf089");

//Obtem id do cliente
String idCliente = WEBTACryptoUtil.getIdFromDecodedFileKey(fileKey);

//Obtem chave de criptografia para a transferência automática
byte[] transfKey = WEBTACryptoUtil.getKeyFromDecodedFileKey(fileKey);

...
```

3.2 WEBTAOutputStream

Esta classe contém os métodos necessários para geração dos arquivos criptografados no formato WebTA. Além de criptografado o conteúdo do arquivo também é compactado. Estes métodos deverão ser inseridos no seu Sistema, no trecho de código onde são gerados os arquivos de remessa. Desta forma estaremos adicionando uma característica de Segurança ao processo, pois os arquivos deixarão de ser gerados em formato texto-aberto. Esta classe é uma especialização da classe OutputStream.

***Pacote**:* `br.com.bradesco.webta.security.crypto.WEBTAOutputStream`

Descrição dos métodos:

3.2.1 **WEBTAOutputStream**

Construtor da classe, cria um objeto de saída de dados criptografados e compactados no formato WebTA.

```
public WEBTAOutputStream(String outputFileName, String directory, byte[] key)
throws ParameterException, GZipException
```

Parâmetros de entrada:

- outputFileName: nome do arquivo onde os dados criptografados de saída serão gravados;
- directory: diretório onde o arquivo final será gerado;
- key: chave para criptografar os dados.

Exceções:

- ParameterException: erro na passagem dos parâmetros;
- GZipException: erro na execução do algoritmo de compactação.

3.2.2 **write**

Grava no arquivo os dados, compactando e criptografando-os. Este método possui duas assinaturas, descritas a seguir:

```
public void write(byte[] b, int off, int len) throws IOException
```

Comprime e criptografa *len* bytes do buffer *b*, iniciando na posição *off* do buffer. Os dados resultantes são gravados no arquivo de saída.

Parâmetros de entrada:

- b: buffer contendo os dados a serem compactados e criptografados;
- off: posição (0 = posição inicial) do buffer a partir da qual o dado será compactado e criptografado;
- len: quantidade de bytes para compactar e criptografar. Este método tratará buffer de dados com até 64 Kbytes por chamada deste método; para quantidade superior a este valor será gerada uma exceção.

Exceção:

IOException: se ocorrer um erro de I/O.

```
public void write(byte[] b) throws IOException
```

Comprime e criptografa todo o conteúdo do buffer *b*, sendo que os dados resultantes são gravados no arquivo de saída.

Parâmetros de entrada:

- b: buffer contendo os dados a serem compactados e criptografados. Este método tratará buffer de dados com até 64 Kbytes por chamada deste método; para buffer com tamanho superior a este valor será gerada uma exceção.

Exceção:
IOException: se ocorrer um erro de I/O.

3.2.3 **size**

Retorna a quantidade de bytes armazenados no objeto de saída.

```
public int size()
```

Retorno:
A quantidade de bytes dos dados armazenados no arquivo/objeto de saída.

3.2.4 **flush**

Executa um *flush* no buffer de saída, forçando que quaisquer bytes armazenados sejam gravados.

```
public void flush() throws IOException
```

Exceção:
IOException: se ocorrer um erro de I/O.

3.2.5 **close**

Executa um *flush* dos dados e fecha o arquivo/objeto de saída.

```
public void close() throws IOException
```

Exceção:
IOException: se ocorrer um erro de I/O.

Exemplo de utilização da classe WEBTAOutputStream:

```
WEBTAOutputStream wos = null;
byte[] bufCripto = new byte[8192];

try{

    //Obtem chave de criptografia
    byte[] chaveCripto = WEBTACryptoUtil.decodeKeyFile(new
       File("c:/CRIPTOGRAFIA200805191017.bin"), "U9dsdfos");


    //Cria objeto para gerar o arquivo criptografado
    wos = new WEBTAOutputStream("COBN1205.REM", "C:/REMESSA",
        chaveCripto);

    //Loop para gerar um arquivo de remessa criptografado
    while (temDados) {
         ...

         //Aplicação processa as informações para gerar o arquivo de remessa
         //e o armazena em bufCripto.

         //Escreve o buffer, criptografando-o
         wos.write(bufCripto);

         ...
    }

} catch (GZipException e) {
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} catch (ParameterException e) {
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} catch (IOException e) {
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} catch (CryptoException e) {
```

```
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} finally {
     if (wos != null) {
        try {
           wos.close();
        } catch (IOException e){
           //System.out.println (e.getMessage());
        }
     }
}
```

3.3 WEBTAInputStream

Esta classe contém os métodos necessários para descriptografar e descomprimir os arquivos no formato WebTA gerados pela classe WEBTAOutputStream. Estes métodos deverão ser inseridos no seu Sistema no trecho de código onde são lidos os arquivos de retorno. Para mantermos a característica de Segurança proposta, é recomendável que durante o processo de descriptografia o conteúdo do arquivo não seja gravado em disco, seja processado diretamente por seu Sistema. Esta classe é uma especialização da classe InputStream.

Pacote: `br.com.bradesco.webta.security.crypto.WEBTAInputStream`

Descrição dos métodos:

3.3.1 WEBTAInputStream

Construtor da classe, cria um objeto de entrada de dados que trata um arquivo criptografado e compactado no formato WebTA, ou seja, descriptografa e descompacta um arquivo gerado pela classe WEBTAOutputStream.

```
public WEBTAInputStream(String inputFileName, String directory,  byte[] key)
throws  ParameterException
```

Parâmetros de entrada:

- inputFileName: nome do arquivo a ser descriptografado;
- directory: diretório onde encontra-se o arquivo;
- key: chave para descriptografar o arquivo.

Exceção:

ParameterException: erro na passagem de parâmetros

3.3.2 read

Lê os dados do arquivo/objeto de entrada, descriptografando e descompactando-os. Este método possui 2 assinaturas, descritas a seguir:

```
public int read(byte[] b, int off, int len) throws IOException
```

Lê o arquivo/objeto de entrada, armazenando os dados descriptografados e descompactados no buffer ***b.***

Parâmetros de entrada:

- off: posição a partir da qual os dados devem ser armazenados no buffer ;
- len: quantidade máxima de bytes que podem ser copiados no buffer.

Parâmetro de saída:

b: buffer para armazenar os dados descriptografados e descompactados.

Retorno:

A quantidade de bytes retornados, equivale ao número de bytes inseridos no buffer *b*.

Exceção:

IOException: erro na passagem de parâmetros

```
public int read(byte[] b) throws IOException
```

Lê o arquivo/objeto de entrada, descriptografa e descompacta os dados. O método detectará o tamanho do buffer (b.length), e preencherá o buffer com no máximo o seu tamanho. Este método é equivalente a chamar “read (b, 0, b.length)”.

Parâmetro de saída:
b: buffer para armazenar os dados descriptografados e descompactados.
Retorno:
A quantidade de bytes efetivamente armazenada em b.
Exceção:
IOException: erro na passagem de parâmetros

3.3.3 **close**

Finaliza o objeto de entrada, fecha o arquivo e libera os recursos alocados pelo objeto.

```
public void close() throws IOException
```

Exceção:
IOException: erro na passagem de parâmetros

Exemplo de utilização da classe WEBTAInputStream:

```
WEBTAInputStream wis = null;
byte[] bufDecripto = null;

try{

   //Obtem chave de criptografia
   byte[] chaveCripto = WEBTACryptoUtil.decodeKeyFile(new
       File("c:/CRIPTOGRAFIA200805191017.bin"), "U9dsdfos");

   //Cria objeto para ler o arquivo criptografado
   wis = new WEBTAInputStream("CB2904100.RET", "C:/RETORNO", chaveCripto);

   //Loop de leitura, onde o conteudo do arquivo
   //é lido, descriptografado, descomprimido e armazenado em bufDecripto.
   //Este loop e´executado enquanto houver dados para leitura
   while(wis.read(bufDecripto) > 0) {

      //...

      //A aplicação processa os dados
      //do arquivo de retorno


      //...
   }

} catch (ParameterException e){
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} catch (IOException e){
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} catch (CryptoException e){
    //Efetua tratamento da excecao
    //System.out.println (e.getMessage());
} finally {
   if (wis != null) {
      try {
          wis.close();
      } catch (IOException e){
          //Efetua tratamento da excecao
          //System.out.println (e.getMessage());
      }
   }
}
```

3.4 CryptoException, ParameterException e GZipException

Pacotes:

```
br.com.bradesco.webta.security.exception.CryptoException
br.com.bradesco.webta.security.exception.GZipException
br.com.bradesco.webta.security.exception.ParameterException
```

Estas são as exceções que devem ser tratadas para a utilização das classes de Criptografia.

4. API de Criptografia - Biblioteca C - Windows

A biblioteca C de Criptografia de Arquivos do WebTA contém as funções necessárias para manipulação dos dados no processo de transmissão/recepção de arquivos criptografados no formato WebTA.

Esta biblioteca é uma DLL para utilização no desenvolvimento de aplicações executadas em ambiente Microsoft Windows 32 bits, dentre os arquivos distribuídos no Kit de Desenvolvimento estão:

- WEBTAEncoderLib.dll: a DLL propriamente dita;
- ZLIB1.dll *: DLL necessária para a compactação dos dados;
- WEBTAEncoderLib.h: arquivo de inclusão, necessário para utilização no desenvolvimento em Linguagem C;
- WEBTAEncoderLib.lib: arquivo para "linkagem" estática, necessário para utilização no desenvolvimento em Linguagem C;
- WEBTAEncoderLib.vb: módulo com declaração das funções para utilização em uma aplicação Visual Basic.

É pré-requisito para sua utilização a instalação prévia do Microsoft Framework .NET 2.0:

http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=pt-br

Para facilitar o entendimento, agrupamos as funções contidas na DLL em:

- Decodificação dos arquivos de chaves: funções para decodificar os arquivos de chaves do cliente;

- Criptografia de arquivos: funções para gerar os arquivos de remessa comprimidos e criptografados no formato WebTA;

- Descriptografia de arquivos: funções para descriptografar e descomprimir os arquivos de retorno criptografados do WebTA.

(*) A ZLIB1.dll é uma DLL gratuita, disponibilizada para quaisquer fins. (http://www.zlib.net)

4.1 Decodificação dos arquivos de chaves

Neste item estão agrupadas as funções necessárias para decodificar os arquivos de chave que foram gerados pelo Master da Empresa no Ambiente Gerencial do site WebTA.

Descrição das funções:

4.1.1 **fDecodeKeyFile**

Este método decodifica o arquivo CRIPTOGRAFIAaaaammddhhMM.bin, que é um dos arquivos gerados no Ambiente Gerencial do WebTA, obtendo a chave necessária para criptografar os arquivos de remessa e descriptografar os arquivos de retorno.

```
int fDecodeKeyFile(char *filepath, char *senha, BYTE *key, char *msgErro)
```

Parâmetros de entrada:

- filepath: caminho completo do arquivo a ser decodificado;
- senha: senha utilizada para abrir o arquivo;

Parâmetros de saída:

- key: chave de criptografia/descriptografia de arquivos. Este parâmetro será utilizado nos métodos de iniciação para criptografia e descriptografia de arquivos. Neste parâmetro deve-se passar um ponteiro para um buffer pré-alocado com o tamanho de 16 bytes.
- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

1 = função executada com sucesso;
0 = erro na execução da função.

Exemplo de utilização da função em linguagem C:

...

```
unsigned char chaveCripto[16];
char mensagemErro[512];

if (!fDecodeKeyFile("E:/criptografia200805191537.bin", "uio32*A",
chaveCripto, mensagemErro)) {
     //Tratar erro
}
```

...

4.1.2 fDecodeKeyFileEx

Este método decodifica o arquivo TRANSFERENCIAaaaammddhhMM.bin, que é um dos arquivos gerados no Ambiente Gerencial do WebTA, obtendo o id do cliente e a chave de criptografia necessários no processo de Autenticação da Transferência Automática de Arquivos.

```
int fDecodeKeyFileEx(char *filepath, char *senha, char *id, BYTE *key, char
*msgErro)
```

Parâmetros de entrada:

- filepath: caminho completo do arquivo a ser decodificado;
- senha: senha utilizada para abrir o arquivo.

Parâmetro de saída:

- id: identificação do usuário, deve-se passar neste parâmetro um ponteiro para um buffer pré-alocado com o tamanho de 21 caracteres;
- key: chave de criptografia, deve-se passar neste parâmetro um ponteiro para um buffer pré-alocado com o tamanho de 16 bytes.
- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

1 = função executada com sucesso;
0 = erro na execução da função.

Exemplo de utilização da função em linguagem C:

...

```
char idCliente[21];
unsigned char chaveTransferencia[16];
char mensagemErro[512];

//Decodifica arquivo que contem parametros
//necessarios para o processo de Autenticacao
//na Transferência Automática de Arquivos
if (!fDecodeKeyFileEx("E:/transferencia200805191537.bin", "au8df9s",
      idCliente, chaveTransferencia, mensagemErro)) {
         //Tratar erro
```

```
        }
...
```

4.2 Criptografia de arquivos

Neste item estão agrupadas as funções necessárias para geração dos arquivos criptografados no formato WebTA. Além de criptografado o conteúdo do arquivo também é compactado. Estes métodos deverão ser inseridos no seu Sistema, no trecho de código onde são gerados os arquivos de remessa. Desta forma estaremos adicionando uma característica de Segurança ao processo, pois os arquivos deixarão de ser gerados em formato texto-aberto.

Descrição das funções:

4.2.1 fInitEncoder

Esta função prepara a API para a geração de um arquivo criptografado.

```
void* fInitEncoder(const char *fileName, const char *directory, const BYTE
*key, char *msgErro)
```

Parâmetros de entrada:

- fileName: nome do arquivo onde os dados criptografados serão gravados;
- directory: diretório onde o arquivo final será gerado;
- key: chave para criptografar os dados.

Parâmetro de saída:

- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

<> NULL: handle para gerar o arquivo criptografado;
= NULL: erro na execução da função.

4.2.2 fWritedata

Grava no arquivo os dados, compactando e criptografando-os.

```
int fWriteData(void *handle, const BYTE *data, const int dataLen, char
*msgErro)
```

Parâmetros de entrada:

- handle: handle retornado pela função fInitEncoder();
- data: dados para serem comprimidos e criptografados;
- dataLen: tamanho dos dados para serem criptografados. Somente será permitido que sejam criptografados até 64 Kbytes por chamada desta função.

Parâmetro de saída:

- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

1 = função executada com sucesso;
0 = erro na execução da função.

4.2.3 fCloseEncoder

Fecha o arquivo de saída e libera os recursos alocados para a criptografia dos dados.

```
int fCloseEncoder(void* handle, char *msgErro)
```

Parâmetros de entrada:

- handle: handle retornado pela função fInitEncoder();

Parâmetro de saída:

- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

1 = função executada com sucesso;
0 = erro na execução da função.

Exemplo de utilização da Criptografia de arquivos em linguagem C:

```
BOOL temDados = TRUE;
unsigned char chaveCripto[16];
void *criptoHandle = NULL;
unsigned char *bufferDados = NULL;
int tamBufferDados = 0;
char mensagemErro[512];



//Decodifica chave para criptografar arquivo
if (!fDecodeKeyFile("E:/criptografia200805191537.bin", "afE8df",
                    chaveCripto, mensagemErro))
{
    printf ("Erro ao decodificar Chave de Criptografia - mensagem de erro:
             %s\n", mensagemErro);
    return -1;
}

//Cria handle para processar a criptografia do arquivo
criptoHandle=fInitEncoder("CB2904100.REM", "C:/REMESSA",chaveCripto,
                          mensagemErro);
if (criptoHandle == NULL)
{
    printf ("Erro na iniciacao da criptografia do arquivo - mensagem de
             erro: %s\n", mensagemErro);
    return -1;
}

//Loop para gerar o arquivo de remessa criptografado
while (temDados)
{
    //...

    //Recebe dados do Sistema
    //Aponta "bufferDados" para os dados a criptografar
    //Inicia "tamBufferDados" com o tamanho dos dados a criptografar

    //Criptografa dados, armazenando-os no arquivo
    if (!fWriteData(criptoHandle, bufferDados, tamBufferDados,
                    mensagemErro))
    {
        printf ("Erro na criptografia dos dados - mensagem de erro: %s\n",
                 mensagemErro);
        fCloseEncoder(criptoHandle, mensagemErro);
        return -1;
    }

    //...
}

//Finaliza processamento de criptografia do arquivo
fCloseEncoder(criptoHandle, mensagemErro);
```

4.3 Descriptografia de arquivos

Neste item estão agrupadas as funções necessárias para descriptografar e descompactar os arquivos codificados no formato WebTA. Estes métodos deverão ser inseridos no seu Sistema no trecho de código onde são lidos os arquivos de retorno. Para mantermos a característica de Segurança proposta, é recomendável que durante o processo de descriptografia o conteúdo do arquivo não seja gravado em disco, seja processado diretamente por seu Sistema.

Descrição das funções:

4.3.1 **fInitDecoder**

Esta função prepara a API para descriptografar um arquivo.

```
void* fInitDecoder(const char *fileName, const char *directory, const BYTE
*key, char *msgErro)
```

Parâmetros de entrada:

- fileName: nome do arquivo a ser descriptografado;
- directory: diretório onde encontra-se o arquivo;
- key: chave para descriptografar o arquivo.

Parâmetro de saída:

- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

<> NULL: handle para ler o arquivo criptografado;
= NULL: erro na execução da função.

4.3.2 **fReadData**

Lê o arquivo/objeto de entrada, armazenando os dados descriptografados e descompactados no buffer *data*.

```
int fReadData(void *handle, BYTE *data, int dataLen, char *msgErro)
```

Parâmetros de entrada:

- handle: handle retornado pela função fInitDecoder();
- dataLen: tamanho do buffer passado como parâmetro.
-

Parâmetro de saída:

- data: buffer onde os dados serão armazenados;
- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

-1: não existem mais dados para leitura;
0: erro na execução da função;
> 0: quantidade de dados lidos e armazenados no buffer.

4.3.3 **fCloseDecoder**

Fecha o arquivo de entrada e libera os recursos alocados para descriptografar os dados.

```
int fCloseDecoder(void* handle, char *msgErro)
```

Parâmetros de entrada:

- handle: handle retornado pela função fInitDecoder();

Parâmetro de saída:

- msgErro: ponteiro para string pré-alocada (com 512 bytes) onde, em caso de erro, será armazenada uma mensagem descritiva do erro.

Retorno:

1 = função executada com sucesso;
0 = erro na execução da função.

Exemplo de utilização da Descriptografia de arquivos:

```
unsigned char chaveCripto[16];
void *deCriptoHandle = NULL;
unsigned char bufferAux[8192];
int qtdeByteLidos = 1;
char mensagemErro[512];


//Decodifica chave para descriptografar arquivo
if (!fDecodeKeyFile("E:/criptografia200805191537.bin", "aSd09w",
                    chaveCripto, mensagemErro))
{
    printf ("Erro ao decodificar Chave de Criptografia - mensagem de erro:
            %s\n", mensagemErro);
    return -1;
}

//Cria handle para descriptografar arquivo recebido do WebTA
//O arquivo CB2904100.RET foi recebido do WebTA e encontra-se
//armazenado em C:/RETORNO
deCriptoHandle = fInitDecoder("CB2904100.RET", "C:/RETORNO", chaveCripto,
                              mensagemErro);
if (deCriptoHandle == NULL)
{
    printf ("Erro na iniciacao da descriptografia do arquivo - mensagem de
             erro: %s\n", mensagemErro);

    return -1;
}

//Enquanto houver dados para descriptografar
//do arquivo de retorno
while(qtdeByteLidos > 0)
{
    //Descriptografa e descomprimi os dados do arquivo
    qtdeByteLidos = fReadData(deCriptoHandle, bufferAux,
                              sizeof(bufferAux), mensagemErro);

    if (qtdeByteLidos > 0)
    {
        //O Sistema processa os dados do arquivo de retorno
        //...
    }

}

//Verifica o que determinou a saída do loop
if (qtdeByteLidos == 0)
{
    //Tratamento de erro, pois houve erro
    //na descriptografia
    printf ("Erro na descriptografia do arquivo - mensagem de
             erro: %s\n", mensagemErro);
}


//Finaliza processamento de descriptografia do arquivo
fCloseDecoder (deCriptoHandle mensagemErro);
```

4.4 Utilização em uma classe .NET

Para a utilização da DLL em uma classe .NET (*managed code*), será necessário efetuar a importação de suas funções utilizando o atributo *DllImport*, e assim invocá-las.

Acessando o *link* abaixo, é possível obter maiores informações de como efetuar este tipo de importação:

http://msdn.microsoft.com/en-us/library/aa288468(VS.71).aspx

Exemplo:

```
[DllImport("WEBTAEncoderLib.dll",
                CharSet = CharSet.Ansi,
                CallingConvention = CallingConvention.Cdecl)]
    public static extern IntPtr fInitEncoder(
        [MarshalAs(UnmanagedType.LPStr)]String filename,
        [MarshalAs(UnmanagedType.LPStr)]String directory,
        byte [] key,
        [MarshalAs(UnmanagedType.LPStr)]StringBuilder msgErro);
```