# SysKeeper-2000 反向隔离装置(单 bit)API 使用手册

南京南瑞集团公司信息系统分公司

# 注意：

本手册中的内容是南瑞安全隔离装置（反向单 bit）API 使用手册。本材料的相关权利归南瑞集团公司信息系统分公司所有。用户手册的任何部分未经本公司许可，不得转印、影印或复印。

# 反向隔离装置（单bit）API使用手册

**Version1.0　2008-2-22**

**南瑞集团公司信息系统分公司**



本资料将定期更新，如预获取最新相关信息，

请访问南瑞集团公司网站：http://www.nari-china.com

您的意见和建议请发送至：sp_support@nari-china.com

**南瑞集团公司信息系统分公司**

南京南瑞路 8 号，210003

电话（TEL）:025-83096601　025-83096605(市场部)

025-83096702　025-83096712(技术支持)

传真（FAX）:025-83096701

# 一、简介

反向隔离装置提供一组 API（Windows 平台），为安全 III 区应用软件提供 E 文件检查、编码转换和数字签名等操作，经过这些操作之后，就可以将转换过的 E 语言文件传输给反向隔离装置。其中的签名操作部分是通过智能 UsbKey 来完成的，因此在进行文件签名和转换操作时，必须插入 UsbKey。

用户使用时需要将我们提供的几个动态链接库 **UsbKey.dll，libeay32.dll，WDSCard.dll，WDAlg.dll，WDCSP.dll，WDcrwv.dll，WDSfkm.dll** 放置在同一个目录内，比如用户应用程序目录或系统目录。提供的函数的接口说明在 UsbKeyLib.h 文件中，该文件中有各个函数的详细使用说明。另外，我们还提供了一个 test 程序供使用者参考。

# 二、UsbKey基本用法

## 2.1 USBKEY的初始化

在使用 UsbKey 之前，必须对 UsbKey 进行初始化，初始化的具体步骤如下：

1) 调用 UsbKey_Init 对 UsbKey 进行格式化，格式化操作主要是删除 Usbkey 中以前的所有文件并重新建立需要的目录和文件；
2) 调用 UsbKey_GenRsa 函数产生一个 RSA 密钥对，并将产生的密钥对写入 UsbKey 中的相应安全文件中去；
3) 调用 UsbKey_MakeReq 函数为刚生成的 RSA 密钥对制作一个证书请求文件（PEM 格式），制作的证书请求文件需要用调度 CA 进行签发（如未建立调度证书系统，也可以参见光盘中的证书签发文档）；
4) 调用 UsbKey_CertImport 函数将签发的证书（PEM 格式）导入到 UsbKey 中去。

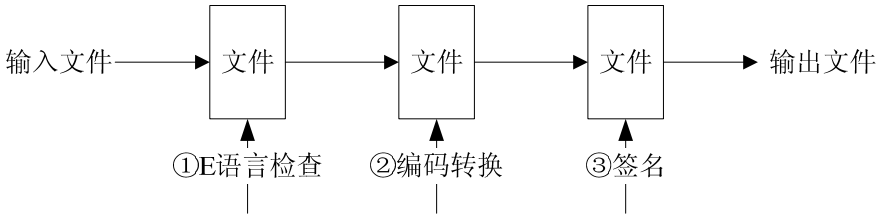
说明：

一般来说，只需要在初次开发的时候对智能UsbKey进行一次格式化即可，对于普通用户，可以屏蔽掉此功能；如果调用了步骤 2 重新生成了RSA密钥对，则必须同时完成步骤 3 和 4。反向隔离装置端或接收端软件在验证文件签名的合法性时也需要用到步骤 3 签发出的证书。

## 2.2 文件操作

### 2.2.1 文件转换

文件转换操作是将待传送的普通文件进行必要的检查、转换和签名操作，生成隔离装置所认可的合法文件。




1) E 语言检查：根据 E 语言的模式文件检查输入的文件是否与模式文件匹配，即输入文件是否是按照模式文件的格式进行编写，此检查不改变文件的内容。(此操作由隔离装置附带的 UsbKey-Tools 工具完成，在第一次进行 E 语言文件传输时，建议使用此工具对要传输的 E 语言文件进行 E 文本合法性验证)。
2) 编码转换：主要是将半角字符转换成全角字符，此转换会改变源文件内容，使原文件变大；
3) 数字签名：用 UsbKey 及提供的 API 函数对文件进行签名操作，并将签名数据附在文件结尾。

**注：如果进行了 2、3 步的操作，在对文件进行恢复时必须按逆向顺序进行。**

在外网的应用软件中可以调用函数 UsbKey_FileConvert 完成文件的转换。说明如下：

1) 如果需要 E 语言检查，则在调用此函数之前需要调用

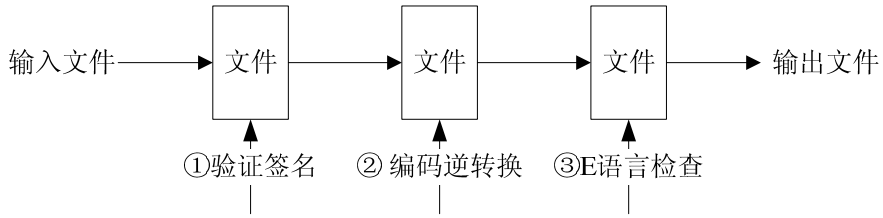
UsbKey_SeteModuleFileDir 函数指定 e 语言模式文件的目录；

2) 进行文件转换时必须插入 UsbKey；

3) 调用此函数之前，必须至少正确验证过一次 pin 码（调用 UsbKey_PinVerify 函数验证 pin 码）。

### 2.2.2 文件恢复

文件恢复是将前面转换过的文件还原,此操作在隔离装置内网单元和内网接收文件的服务端进行。




1) 验证签名：反向隔离装置用 UsbKey 的文件证书对接收到的文件进行解签名操作，如果验证签名通过，则将此文件传输给内网文件接收端，否则将文件丢弃；

2) 编码逆转换：将纯文本文件转换成文本文件，主要是将全角字符转换成半角字符，此转换会改变源文件内容，使原文件变小；

3) E 语言检查：反向隔离装置根据 E 语言的模式文件检查输入的文件是否与装置内的模式文件匹配，即输入文件是否是按照模式文件的格式来些的。只有符合配置的 E 模板文件才容许传输到内网。

在内网的应用软件中可以调用函数 UsbKey_FileRecover 函数完成文件的恢复操作。

说明如下：

1) 此功能仅做测试使用，因为一般文件的恢复操作在装置端或文件的接收端进行；

2) 此功能无需 usbkey，但需要用到 usbkey 的证书进行验证签名的操作，在调用次函数之前需要调用函数 UsbKey_SetCertFileDir 指定证书的目

录。

## 2.3 其它操作

### 2.3.1 验证/修改pin码

UsbKey 的许多操作需要正确验证了 pin 码才能进行（正确验证了 pin 码才有权限对 UsbKey 进行操作）。

调用 UsbKey_PinVerify 函数进行 pin 码验证；

调用 UsbKey_PinModify 函数对 pin 码进行修改。

说明：

注意 pin 码的长度为 3～8 位；

Pin 码的验证尝试次数最多为 6 次，如果 6 次的 pin 码均错误，则 UsbKey 将死锁；错误数会在卡内累计，当正确验证了 pin 码之后，错误累计数清零。

如果 UsbKey 死锁，则必须调用函数 UsbKey_Init 对 UsbKey 重新进行格式化。并程序生成 RSA，制作证书请求等（参见 UsbKey 的初始化部分）。

### 2.3.2 导出证书

如果 UsbKey 内有证书，则可以调用 UsbKey_CertExport 函数导出（PEM 格式）。

关于 UsbKey 的具体使用请参考我们提供的例子程序（VS2002 源码）

# 三、Usb Key API使用说明

1) char * WINAPI UsbKey_GetLastErr()

/*****************************************************************

取得UsbKey的最近一次错误信息

如果调用某个UsbKey的函数返回错误，要想得到具体错误信息，应立即调用此函数

如果某个函数的返回值为-1，必须调用此函数才能获得具体错误信息

@Params

* 无

@Return

* 返回字符串指针

**********/

2) 1.2 int WINAPI UsbKey_Init

(

const char *pin,

int pinLenonst,

const char *ubPin,

int ubPinLen

)

/**********

UsbKey的初始化函数

UsbKey的使用步骤：

1.调用UsbKey_Init函数初始化UsbKey

2.调用UsbKey_GenRsa产生RSA

3.调用UsbKey_MakeReq生成证书请求文件。生成的证书请求文件要用openssl来签发

4.调用UsbKey_CertImport函数，将签发过的证书导入到UsbKey当中（证书还需用在其它需要的地方）

只有完成了上述3步，才能正常使用UsbKey

@Params

* pin: in 设置的初始pin码

* pinLen: in pin码的长度（pin码长度必须在3～8位之间，下同）

```
        * ubPin:       in 解锁pin码
        * ubPinLen:    in 解锁pin码长度
    @Return
        * ERR_OK: 成功
        * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
看）
*****************************************************************************/
```

3) int WINAPI UsbKey_GenRsa()

```
/*****************************************************************************
    产生UsbKey的rsa密钥对
    见UsbKey_Init函数说明
    @Params
        * 无
    @Return
        * ERR_OK: 成功
        * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
看）
*****************************************************************************/
```

4) int WINAPI UsbKey_MakeReq

```
(
const char *com_name,const
char *local_name,
const char *org_name,
char *dataout,
int *outlen
)
```

/*****************************************************************************

制作证书函数

见UsbKey_Init函数说明

@Params

* com_name:  in 证书请求的主体名

* local_name: in 证书请求的地名

* org_name:  in 证书请求的组织名

* dataout:  in 存储返回证书请求数据(pem格式)的buffer(需要保证至少2k的空间大小)

* outlen: out 返回的数据的长度

@Return

* ERR_OK: 成功

* other:  失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）

*****************************************************************************/

5) int WINAPI UsbKey_MakeReq2

(

const char *com_name,

const char *local_name,

const char *org_name,

char *filename

)

/*****************************************************************************

制作证书函数

```
    见UsbKey_Init函数说明
    @Params
       * com_name:   in 证书请求的主体名
       * local_name: in 证书请求的地名
       * org_name:   in 证书请求的组织名
       * filename:   in 证书请求将保存到的文件
    @Return
       * ERR_OK:  成功
       * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
看）
   ***************************************************************************/
```

6) int WINAPI UsbKey_CertImport
(
const char *certfile
)

```
   /***************************************************************************
    导入证书到UsbKey中
    见UsbKey_Init函数说明
    @Params
       * certfile:   in 证书路径(最好使用绝对路径,相对路径有时会出错)
    @Return
       * ERR_OK:  成功
       * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
看）
   ***************************************************************************/
```

7) int WINAPI UsbKey_CertExport
(

```
    char *dataout,
    int *outlen
    )
/*****************************************************************************
 从UsbKey中导出证书
 @Params
 * dataout:    out 存放导出的数据(需要保证2k的内存大小)
 * outlen:     out 实际导出的数据的长度
 @Return
 * ERR_OK: 成功
 * other:  失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）
*****************************************************************************/
```

8) int WINAPI UsbKey_PinVerify

```
    (
    const char *pin,
    int pinLen
    )
/*****************************************************************************
 验证UsbKey的pin码
 注：许多UsbKey的操作需要正确验证过pin码才能执行
 @Params
    * pin:     in pin码
    * pinLen:  in pin码的长度（pin码长度必须在3～8位之间）
 @Return
    * ERR_OK:  成功
    * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
```

看）

**********/

9) int WINAPI UsbKey_PinModify

(

const char *pin,

int pinLen,

const char *newpin,

int newpinLen

)

/**********

修改UsbKey的pin码

@Params

* pin:  in 旧pin码

* pinLen:  in 旧pin码的长度（pin码长度必须在3～8位之间）

* newpin:  in 新pin码

* newpinLen:  in 新pin码长度（pin码长度必须在3～8位之间）

@Return

* ERR_OK:  成功

* other:  失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）

**********/

10) int WINAPI UsbKey_UnblockPin

(

char *unblockpin,

int ubpinLen,

char *newpin,

```
    int newpinLen
    )
/*****************************************************************************
 解锁被锁定的UsbKey的pin码
 @Params
 * unblockpin:    in 解锁pin码
 * ubpinLen:      in 解锁pin码的长度（解锁pin码长度必须在3～8位之间）
 * newpin:        in 新pin码
 * newpinLen:     in 新pin码长度（pin码长度必须在3～8位之间）
 @Return
 * ERR_OK: 成功
 * other:  失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）
*****************************************************************************/
11) int WINAPI UsbKey_SeteModuleFileDir
    (
    char *dirpath
    )
/*****************************************************************************
 设置e文本的模式文件的目录
 注：必须设置了模式文件的目录后才能进行e文本的检查
 @Params
   * dirpath: in e文本模式文件的目录（注：必须传入绝对路径）
 @Return
   * ERR_OK:  成功
   * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查
```

看）

```
*****************************************************************************/
```

12) int WINAPI UsbKey_SetCertFileDir

```
    (
    char *dirpath
    )
/*****************************************************************************
```

设置证书文件的目录

注：必须设置证书文件的目录后才能进行文件的恢复操作

@Params

* dirpath:  in 存放证书文件的目录（注：必须传入绝对路径）

@Return

* ERR_OK: 成功

* other:  失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）

```
*****************************************************************************/
```

13) int WINAPI UsbKey_FileConvert

```
    (
    char *srcFile,
    char *dstFile,
    bool bEfile,
    bool bTxt,
    bool bSignature
    )

/*****************************************************************************
```

将普通文件进行转换加密操作

@Params

* srcFile: in 源文件路径

* dstFile: in 目的文件路径

* bEfile: in 是否进行e文本检查

* bTxt: in 是否进行编码转换

* bSignature: in 是否对文件进行签名

@Return

* ERR_OK: 成功

* other: 失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）

**********/

14) int WINAPI UsbKey_FileRecover

```
(
char *srcFile,
char *dstFile,
bool bEfile,
bool bTxt,
bool bSignature
)
```

/**********

将转换过的文件还原

@Params

* srcFile: in 源文件路径

* dstFile: in 目的文件路径

* bEfile: in 是否进行e文本检查

* bTxt: in 是否进行编码转换

```
        * bSignature: in 是否对文件进行解签名
    @Return
        * ERR_OK:  成功
        * other:   失败（具体原因见错误码或调用UsbKey_GetLastErr函数查看）
    ********************************************************************************/
```

# 四、附录

## 4.1 USBKEY的错误码

```
    // 错误码
    #define ERR_OK                      0           // 成功
    #define ERR_WRONG_PARAMS            1           // 参数错误
    #define ERR_NEED_VERRIFY_PIN        2           // 需要验证UsbKey的pin码
    // 智能卡操作错误码
    #define WD_OK                               0
    #define WD_USER_CANCEL                      1
    #define WD_OUTOFMEMORY                      0x10000
    #define WD_CANNOT_ESTABLISH_CONTEXT         0x10001
    #define WD_CANNOT_CONNECT_CARD              0x10002
    #define WD_CANNOT_OPEN_REGKEY               0x10003
    #define WD_NO_SCREADER                      0x10004
    #define WD_NO_CARD_IN_READER                0x10005
    #define WD_ERROR_EXTERNAL_AUTH              0x10006
    #define WD_ERROR_VERIFY_SO_PIN              0x10007
```

```
#define WD_ERROR_VERIFY_USER_PIN        0x10008
#define WD_ERROR_INSTALL_USER_PIN       0x10009
#define WD_ERROR_WRITE_FILE             0x1000a
#define WD_ERROR_WAIT_SIGNALED_STATE    0x1000b
#define WD_INVALID_HANDLE_VALUE         0x1000c
#define WD_ERROR_READ_FILE              0x1000d
#define WD_WRONG_PARAMETER              0x1000e
#define WD_ERROR_ICC_RESET              0x1000f
#define WD_ERROR_SELECT_FILE            0x10010
#define WD_ERROR_GET_RANDOM             0x10012
#define WD_ERROR_SECURITY_STATE         0x10013
#define WD_ERROR_RSASIGN                0x10014
#define WD_ERROR_RSADECRYPT             0x10015
#define WD_ERROR_RSAVERIFY              0x10016
#define WD_ERROR_CREATE_FILE            0x10017
#define WD_ERROR_UPDATE_FILE            0x10018
#define WD_ERROR_SET_PROTOCOL           0x10019
#define WD_ERROR_CREATE_KEY             0x1001A
#define WD_ERROR_WRITE_KEY              0x1001B
#define WD_ERROR_CARD_RESET             0x1001C
#define WD_ERROR_ERASE_FILE             0x1001D
#define WD_ERROR_AUTH                   0x1001E
// xxxx表示标准的卡返回的状态码
#define WD_SCARD_RETURN                 0x2xxxx
// 卡返回的标准错误码
```

返回值　　代码说明.　　意义

0xFFFF: "Communication error" 通讯错误

0x9000: "OK" 操作正确

0x6200: "No card or time out or invalid NAD or PIN" 无卡

0x6281: "Reading error/data error" 读错误

0x6282: "End of file" 文件结束

0x6300: "Error in PIN check" 错误的PIN

0x63Cx: "incorrect PIN,x retries possible" x次错误PIN

0x6400: "Reset not successful" 复位不成功

0x6500: "EDC or write error" 校验错

0x6581: "Update not successful" 修改数据不成功

0x6700: "Wrong length Lc" 错误的命令长度

0x6900: "Wrong state" 错误的状态

0x6981: "Wrong file type PUK point" 错误的文件类别

0x6982: "Permittion deny" 权限不够

0x6983: "No retry possible" 不可再试

0x6985: "file exists already" 文件已存在

0x6A00: "Wrong P1/P2 or file not find" 错误的P1/P2

0x6A80: "Wrong parameters command" 错误的参数

0x6A81: "Wrong value for P2" 错误的P2

0x6A82: "File not found" 文件没找到

0x6A84: "No enough memory in file" 文件无足够空间

0x6A86: "Wrong parameters" 错误的参数

0x6B00: "Wrong offset" 错误的偏移量

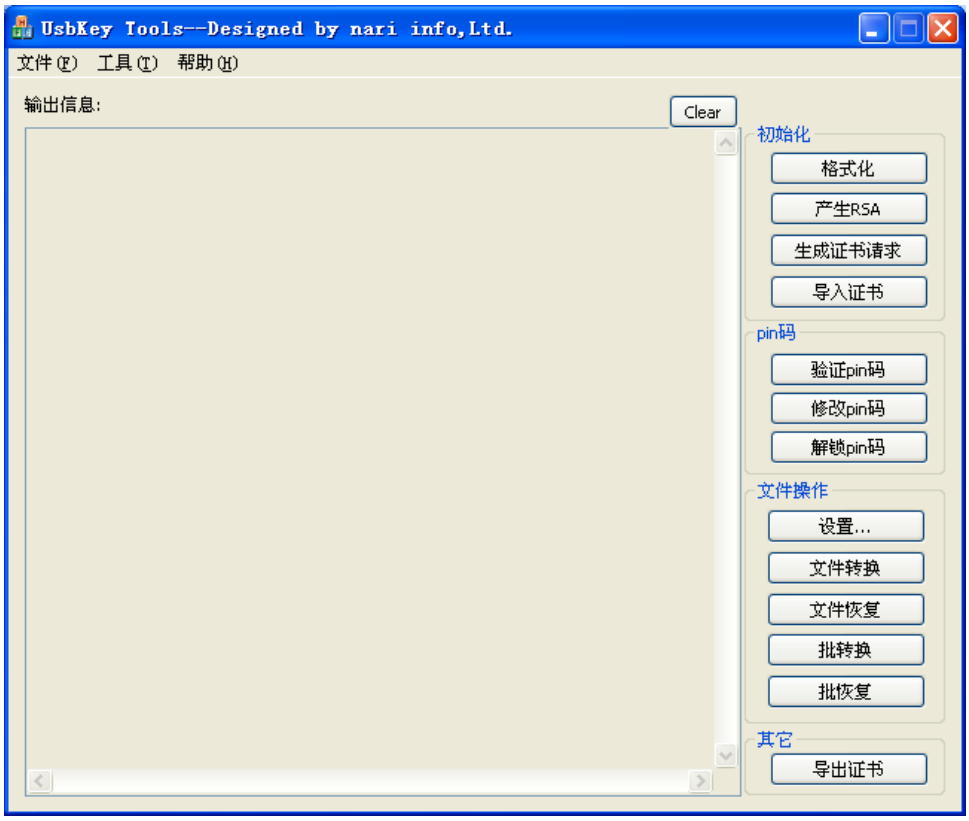
0x6D00: "Invalid instruction code" 无效的指令代码

0x6E00: "Invalid Class Byte" 无效的CLA

0x6FF0: "System error" 系统错误

## 4.2 USBKEY-TOOLS使用手册

UsbKey-Tools是南瑞信息系统分公司提供的一个如何调用NRUsbKey.dll的一个windows下的例子程序。包含了UsbKey的初始化操作、E语言检查、编码转换，文件签名/解签名等基本操作。程序界面下图所示：



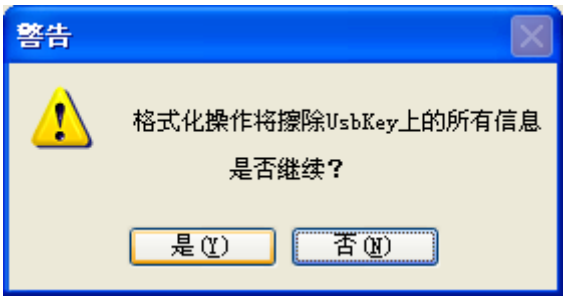

### 4.2.1UsbKey初始化

在使用UsbKey之前，必须对其进行初始化操作。在对UsbKey进行了初始化之后，以后就不需要再进行初始化了（除非需要重新产生RSA，见下面的说明）。

初始化的具体步骤如下：

### 1） 格式化

格式化主要是删除UsbKey原有的目录及各种内部文件，重新划分目录和文件结构，为产生RSA做准备。用户点击了格式化按钮后，将弹出警告信息，如果用户点击“是”，则将对UsbKey进行格式化测试程序设置的初始pin码是123456。

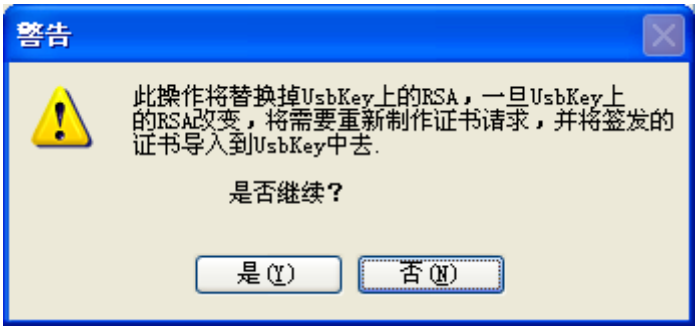



格式化成功后，界面将显示如下信息：

```
Format UsbKey ok,default pin is 123456
```

### 2） 产生 RSA

对UsbKey进行了格式化之后，就可以产生RSA了，点击产生RSA之后，将弹出警告信息，当用户点击了“是”按钮之后，UsbKey内部将生成RSA密钥对，私钥存储到内部的私钥文件中去（不可读），公钥存储到公钥文件中去（可读，公钥需要用来制作证书请求，见1.3 生成证书请求）。




产生RSA需要一定的时间（10秒左右），产生RSA成功后，界面将显示成功信息：

```
Generate RSA ok
```

### 3） 生成证书请求

每一次重新产生RSA之后，都需要重新生成证书请求文件，并对证书请求文件进行签发，生成新的证书（关于如何签发证书，请参看我们提供的相关文档）。

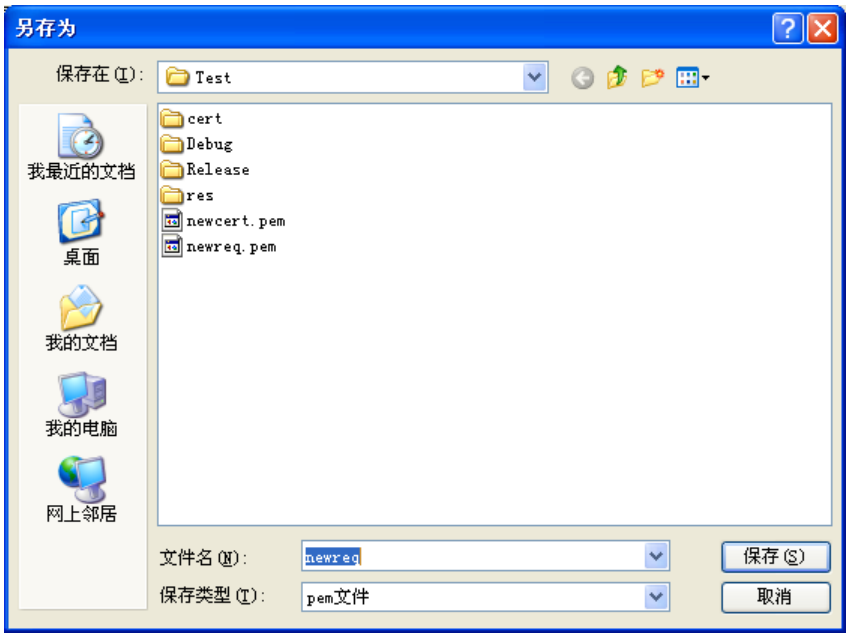
如果您的证书丢失了，您可以点击“生成证书请求”按钮重新生成证书请求文件，并进行签发，而无需重新生成RSA。

签发成的证书文件必须要导入UsbKey中去（见1.4 导入证书），另外证书还需要导入到反向隔离装置中去。

点击“生成证书请求”按钮后，将弹出对话框，用户需要填写证书请求文件所需要的一些信息，这些信息将被记录到证书请求文件中去，同时也将存在于最终签发的证书当中。

**这里需要注意的是，用户在填写证书请求信息时，必须保证 CommonName 的唯一性，即用户在这里所填写的 CommonName 不可以与其它正在使用的证书的 CommonName 一样，也就是说，要保证导入反向隔离装置的所有的证书的 CommonName 不能一样。**


填写完证书请求信息并点击了“确定”按钮后，将弹出保存对话框提示用户保存生成的证书请求文件。

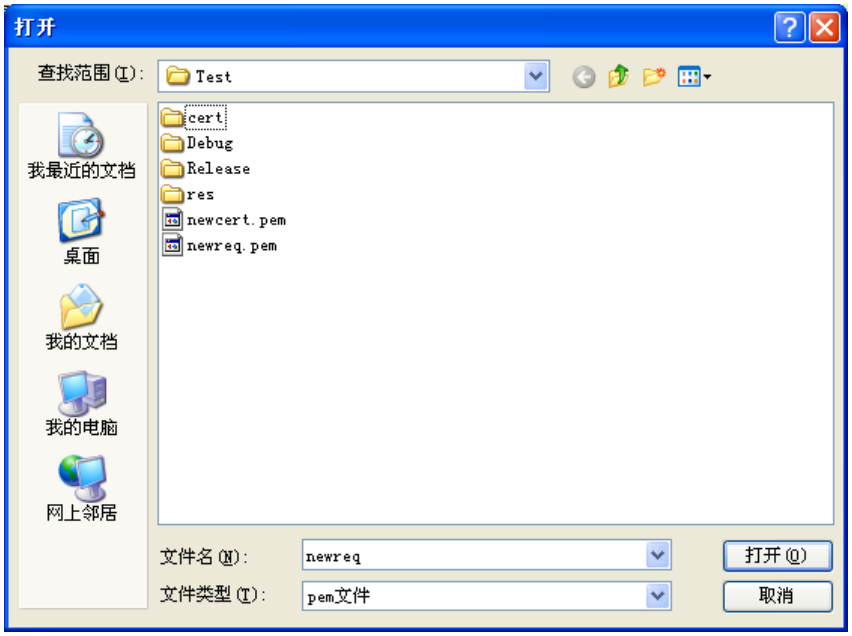



说明：如果**生成证书请求**是紧接着**格式化**和**产生 RSA** 之后做的，则不需要用户验证 pin 码，否则用户需要成功验证 pin 码之后，才能完成此操作。

4） **导入证书**

用户在对产生的证书请求文件进行签发，生成了证书文件之后，不仅需要将证书文件导入反向隔离装置，还需要将证书文件导入 UsbKey 中去。

点击“导入证书”按钮之后，将弹出选择文件对话框，提示用户选择要导入的证书文件，然后回将用户选择的证书文件导入到 UsbKey 中去。

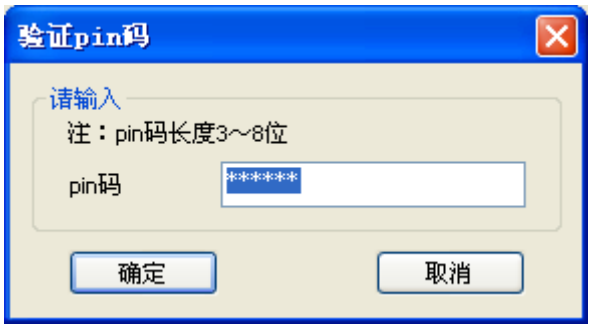


### 4.2.2 pin码

在每次访问 UsbKey 之前，都需要正确验证了 pin 码之后，才能拥有访问权限。当累计 6 次输错 pin 码后，UsbKey 将被锁定，此时只有通过解锁 pin 码来对 UsbKey 进行解锁，并重新设置 pin 码。

1） **验证 pin 码**

用户只有在正确验证了 pin 码之后，才能得到 UsbKey 的访问权限，点击“验证 pin 码”按钮后，将弹出对话框提示用户输入 pin 码，pin 码验证成功后，界面将显示相关成功的信息。

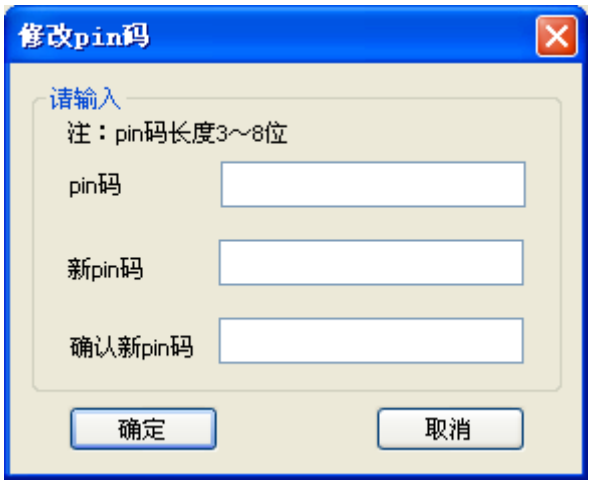



注：测试程序在 UsbKey 初始化时设置的初始 pin 码是 123456，如果用户想更改此设置，在调用我们提供的 dll 库函数时，传入相应的参数即可（具体请参考 UsbKey API 手册中的 API 函数说明）。

2） **修改 pin 码**

用户点击“修改 pin 码“按钮后，将弹出修改 pin 码对话框，当修改成功后，界面将回显示相应的成功信息。

需要注意的是：**pin 码的长度范围在 3～8 位之间**




3） 解锁 pin 码

当 pin 码被锁死后，可通过解锁 pin 码对 UsbKey 进行解锁，并重新设置 pin 码。点击“解锁 pin 码：按钮后，将弹出解锁 pin 码对话框，解锁成功后，界面将显示相关成功的信息。


注：测试程序在 UsbKey 进行初始化时设置的初始解锁 pin 码是 123456，解锁 pin 码只能在 UsbKey 初始化时进行设置，一旦设置就不可更改，除非重新初始化 UsbKey。如果用户想更改此设置，在调用我们提供的 dll 库函数时，传入相应的参数即可（具体请参考 UsbKey API 手册中的 API 函数说明）。

请妥善保管好解锁 pin 码，如果解锁 pin 码被锁死，UsbKey 将被永久锁死，只能更换了。

### 4.2.3 文件操作

文件操作包括操作前的必要设置，E 语言检查，编码转换/逆转换，签名/验证签名等。几种文件操作的基本概念如下：

1．E 语言检查：根据模式文件来检查输入文件是否与相应的模式文件匹配，即输入文件是否按照模式文件来写的；

2．编码转换/逆转换：即半角字符与全角字符之间的相互转换；

3．签名/验证签名：签名是用 UsbKey 的私钥对输入文件进行签名操作，并将签名的数据附在输入文件末尾。验证签名是用证书来验证文件的签名是否合法，如果合法，则去掉文件末尾的签名数据。

**1） 设置**

在进行各种文件操作之前，需要进行一些必要的设置。点击“设置”按钮，将弹出设置对话框


说明：

**模式文件目录**：E 语言检查需要；

**证书文件目录**：文件恢复时，验证签名需要；

**转换目录、转换后存储目录**：批转换时需要；

**恢复目录，恢复后存储目录**：批恢复时需要；

**E 语言检查**：指示是否需要进行 e 语言检查；

**编码转换/逆转换**：指示是否需要半角与全角之间的相互转换，对文件转换操作是半角→全角，对文件恢复操作是全角→半角；

**签名/验证签名**：对文件之后操作指示是否需要进行签名，对文件恢复操作指示是否需要验证签名。

2） 文件转换

对单个文件进行转换，转换的具体操作内容由 **设置** 决定，点击“文件转换”按钮后，将弹出选择文件对话框，用户选择了文件之后，程序将根据 **设置** 中的选项对文件进行相应的操作。

注：如果在 **设置** 中设置了签名/验证签名选项，则在进行文件转换之前需要插入 UsbKey，并且正确验证过 pin 码。

3） 文件恢复

对单个文件进行恢复，恢复的具体操作内容由 **设置** 决定，点击“文件恢复”按钮后，将弹出选择文件对话框，用户选择了文件之后，程序将根据 **设置** 中的选项对文件进行相应的操作。

注：转换时对文件做了哪些操作，恢复时应选择相应的操作选项，否则，恢复的文件不一定是原来的文件了。

在对文件进行编码转换后，再逆转换回去（编码逆转换）的时候，得到的文件不一定与原文件完全相同，这是因为在做编码逆转换的时候，将把文件中所有的全角字符全部转换成半角字符，如果原始文件中含有全角字符，则经编码转换，再编码逆转换后得到的将是半角字符，而非全角字符。

**批转换**

批转换是一次转换一批文件，转换的目录和转换后存储的目录由 设置 中设定，将会扫描转换目录的子目录。点击“批转换”按钮后，程序将根据 设置 中

相应选项进行文件批转换，并将转换的信息显示在界面上。

注：如果在设置 中设置了签名/验证签名选项，则在进行文件转换之前需要插入 UsbKey，并且正确验证过 pin 码。

**4） 批恢复**

批恢复是一次恢复一批文件，待恢复的目录和恢复后存储的目录由 设置 中设定，将会扫描待恢复目录的子目录。点击“批恢复”按钮后，程序将根据**设置**中相应选项进行文件批恢复，并将转换的信息显示在界面上。

### 4.2.5 其它

**1） 导出证书**

导出证书是导出存储在 UsbKey 里的证书，必须保证 UsbKey 里有证书才能导出。点击“导出证书”按钮后，如果导出成功，将弹出保存对话框提示用户保存导出的证书。

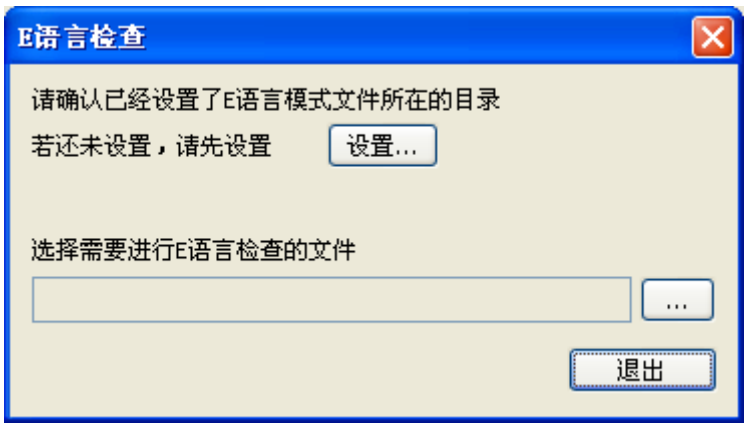
注：在导出证书之前，必须插入 UsbKey 并正确验证过 pin 码。

**2） 工具菜单**

测试程序的“工具”菜单中提供了 E 语言检查、编码转换/逆转换，签名/验证签名等三个独立操作的菜单，方便用户对文件进行操作。

A、 E 语言检查

点击“工具”菜单里的“E 语言检查”，将弹出 E 语言检查对话框，如下图所示：



说明：

进行 E 语言检查之前，需要在 **设置** 对话框里设置 E 语言模式文件。

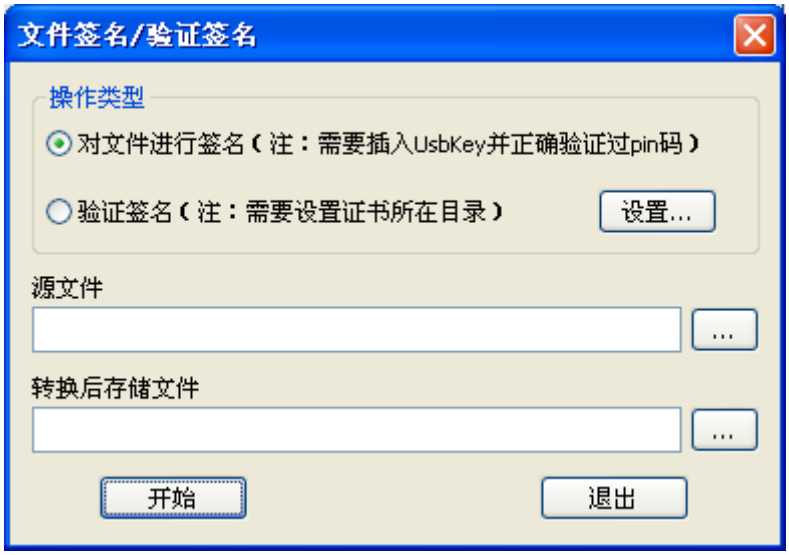
B、编码转换/逆转换

点击“工具”菜单里的“编码转换”菜单，将弹出编码转换对话框，如下图所示：


选择相应的操作类型，并选择了源文件和转换后文件的存储路径后，点击开始按钮，将进行相应的转换操作。

C、签名/验证签名

点击“工具”菜单里的“文件签名”菜单，将弹出签名/验证签名对话框，如下图所示：



如果进行文件签名操作，则需要传入 UsbKey，并正确验证过 pin 码，如果进行验证签名操作，则需要则 **设置** 对话框里设置证书文件所在目录。