

ET199Auto 参考资料



版权所有©2008 坚石诚信科技有限公司

<http://www.jansh.com.cn>

目 录

| | |
|-------------------------------------|----|
| 一、关于ET199Auto..... | 3 |
| 二、ET199Auto的优越性..... | 4 |
| 三、使用说明..... | 5 |
| 3.1 安装..... | 5 |
| 3.2 卸载..... | 5 |
| 3.3 USBKey管理工具使用 | 5 |
| 3.4 申请数字证书 | 8 |
| 四、CAPI接口调用ET199Auto | 9 |
| 4.1 枚举ET199Auto硬件中的证书 | 9 |
| 4.2 使用CAPI接口验证用户PIN（User PIN） | 14 |
| 4.3 签名和验签 | 15 |
| 4.4 RSA加解密 | 22 |
| 五、ET199Auto参数..... | 25 |

一、关于 ET199Auto

ET199Auto是一款无驱无软，采用16位高强度智能卡芯片的高安全性USB Key产品。彻底废除安装光盘，使用更加简易方便。ET199Auto采用无驱设计，功能强大，价格实在，能够使用在各种领域和众多的用途中。特别在网银应用中，使广大银行用户能够最安全，最方便，最简单的通过互联网进行各种操作。ET199Auto将安全性和方便性有效的结合到一起，提高网银安全性的同时，让用户使用更加方便、简洁。

ET199Auto可以做为数字证书的安全载体，敏感数据都被安全地保存在ET199Auto的安全存储区域中，未授权用户是无法接触到这些信息的。数据的签名和加密操作全部在ET199Auto内部完成，私钥从生成的时刻起就一直保存其中，可有效的杜绝黑客程序的攻击。ET199Auto的安全性还在于其使用的加密算法都是被广泛公开，业界公认的，经受了多年考验的标准算法。同时，一流的芯片封装工艺也保证了芯片内数据的安全性。

ET199Auto能够硬件产生512，1024和2048位的RSA密钥对，硬件实现RSA的各种运算。其采用16位的CPU，使用高速无驱的通讯技术，能够十分迅速的完成各种PKI应用的操作。

ET199Auto 提供符合业界广泛认可的 **Microsoft CryptoAPI** 标准接口。任何兼容这种接口的应用程序，都可以立即集成 ET199Auto 进行使用。同时，ET199Auto 也针对多个第三方的软件产品进行了兼容性优化。此外，ET199Auto 内置大容量的安全存储器，可以同时存储多个数字证书。

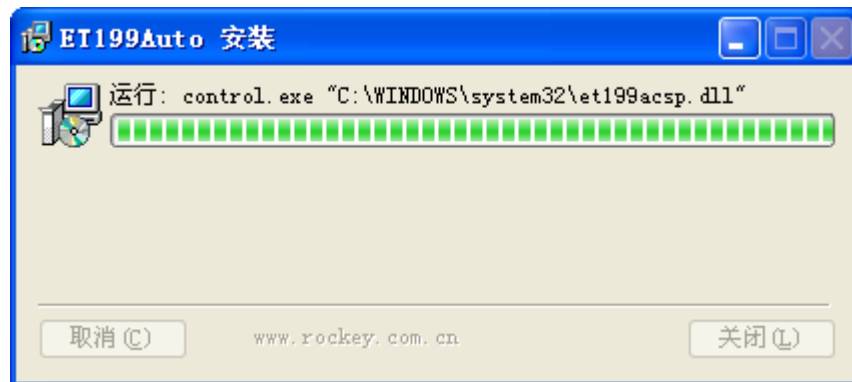
二、ET199Auto 的优越性

- 真正无需安装，即插即用，无驱无软设计，彻底废除安装光盘。
- 采用国外进口 16 位高强度智能卡安全芯片。
- 16K 用户可用空间。
- 中文/英文/繁体中文自适应软件自动安装。
- 安装时需要管理员权限，并有界面提示。使用时不受管理员权限限制，即非管理员亦可使用。
- 提供更新锁内 ISO 文件工具。
- 完善的用户 PIN 码远程解锁方案。
- 高速无驱设备，将高速和应用简便集成到一体。
- 硬件级通讯加密，保证了传输数据的安全性。
- 支持 Microsoft CryptoAPI 标准接口。
- 硬件全球唯一 64 位序列号。
- 用做数字证书的安全载体，可以同时存储多个数字证书。
- 硬件实现 512、1024、2048 位的 RSA 密钥对产生和运算功能。
- 支持 MD5、SHA1 散列算法，DES、3DES 加解密算法。
- 提供多种开发语言示例。
- 支持 WINDOWS 2000/XP/Server 2003/Vista/2008(包括 32 位和 64 位)及各补丁版本。
- 支持 X64 位操作系统。
- 硬件擦写次数 10 万次，保存 10 年。
- 工作温度：0℃—70℃。

三、使用说明

3.1 安装

将 ET199Auto 插入计算机的 USB 接口后，这时机器将会自动运行安装过程，如下图显示：



注意：

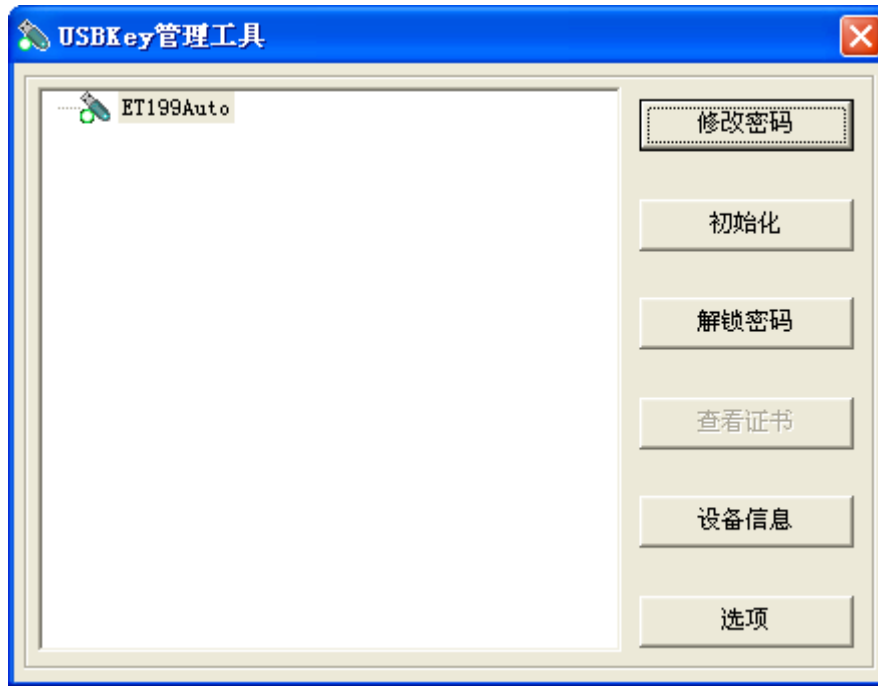
- (1) 如果计算机禁止了光盘自动运行，那么请到“我的电脑”中运行新出现光盘中的 ET199AUTO.EXE 文件，进行手动安装。
- (2) 安装时要是计算机操作系统的管理员身份。在安装过程中遇到杀毒软件或者防火墙提示窗口时，请选择允许操作。
- (3) ET199Auto 可以在操作系统管理员和非管理员权限下使用。

3.2 卸载

选择“开始”→“程序”→“EnterSafe”→“ET199Auto”→“卸载 ET199Auto”进行卸载。

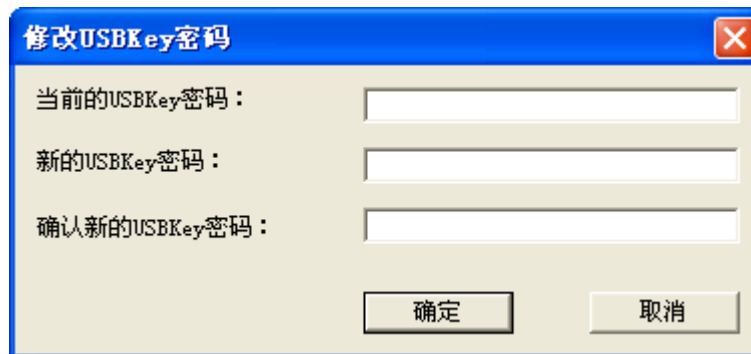
3.3 USBKey 管理工具使用

选择“开始”→“程序”→“EnterSafe”→“ET199Auto”→“管理工具”启动 USBKey 管理工具，如下图所示：



(1) 修改 USBKey 密码

点击“修改密码”按钮，修改 USBKey 密码。**ET199Auto 的默认密码为：1234**。如下图所示：

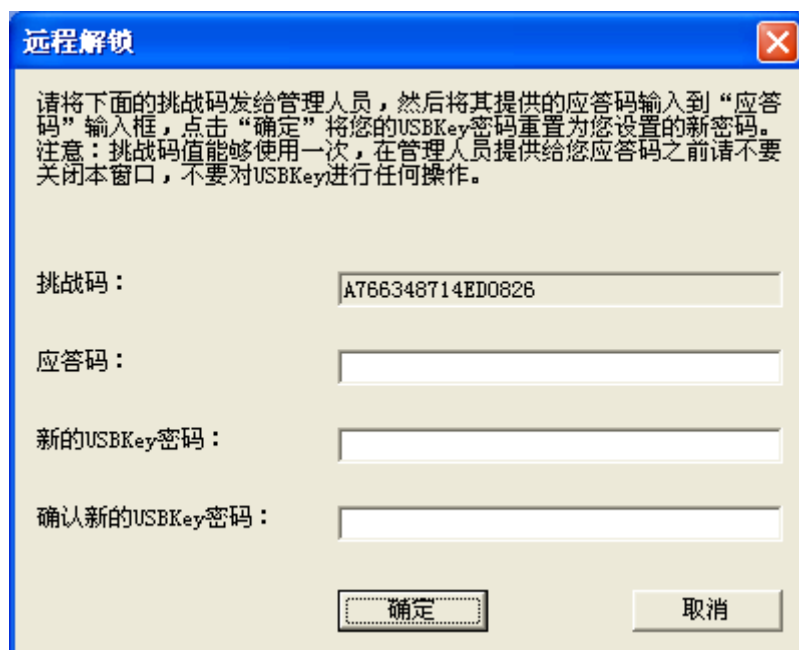


(2) 初始化

点击“初始化”按钮，对锁进行初始化操作。初始化后 ET199Auto 中的数据都将会被删除，且 USBKey 密码恢复为默认的：1234。

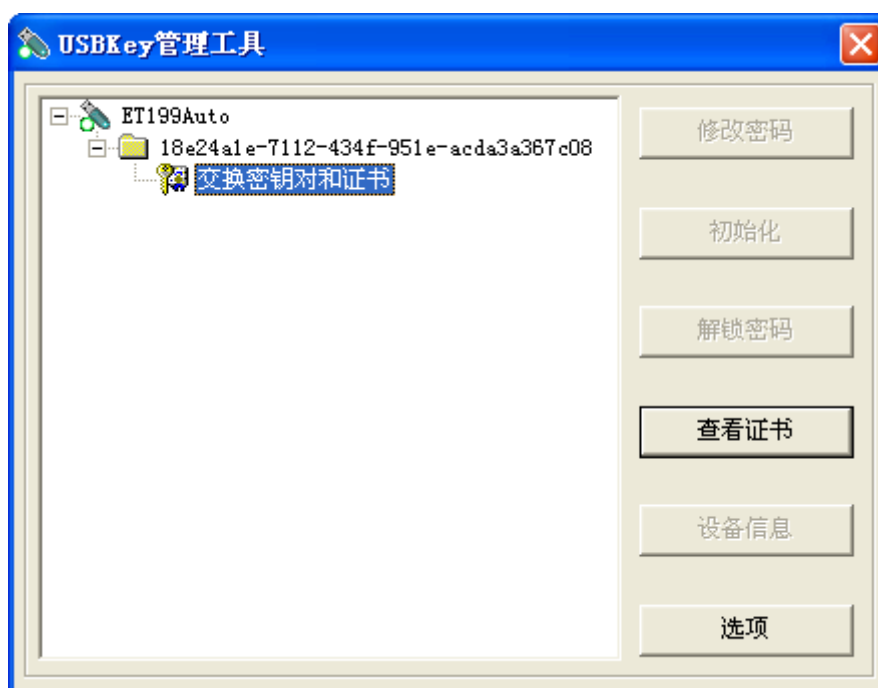
(3) 解锁密码

USBKey 密码的重试次数为 6 次，当连续 6 次输错密码后，USBKey 密码将会被锁死，这时需要与管理员联系进行解锁。点击“解锁密码”按钮，出现解锁窗口，如下图所示。每次弹出解锁窗口时都会有不同的“挑战码”，将“挑战码”发给管理员得到“应答码”，然后填入新的 USBKey 密码，进行解锁。注意：在得到“应答码”之前，不要关闭解锁窗口。



(4) 查看证书

当选择到“交换密钥对和证书”时，可以点击“查看证书”按钮，查看证书信息。如下图所示：



(5) 设备信息

点击“设备信息”按钮，来查看 ET199Auto 设备的一些相关信息

(6) 选项

点击“选项”按钮，可以设置计算机插入 ET199Auto 后是否自动访问网站，并填入网站的地址。

3.4 申请数字证书

ET199Auto 的 CSP 名称为“EnterSafe ET199Auto CSP V1.0”，在 CA 申请证书选择 CSP 时请选择这个 CSP，如下图所示。选择完 CSP 后，点击下一步，输入 USBKey 密码，确定后，ET199Auto 完成产生 RSA 密钥对，并向 CA 中心提交证书申请等操作，成功后显示“安装证书”，成功安装后即完成申请证书过程。

☒ 创建新密钥集

☐ 使用现存的密钥集

CSP:

EnterSafe ET199Auto CSP V1.0

密钥用法: ☒ 交换

密钥大小:

1024

最小值: 512
最大值: 2048

(一般密钥大小: [512](#) [1024](#) [2048](#))

四、CAPI 接口调用 ET199Auto

在使用微软的 CAPI 接口进行开发时应注意，有些定义在微软每年发布的 PSDK 中，因此需要先安装 PSDK。下面的连接是 PSDK2003 的下载地址：

<http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdk-full.htm>

下载并安装后，需要在 VC 中进行配置。在这里我们假设把 PSDK 安装到 C:\Program Files\Microsoft SDK 目录下，在 VC6 中配置为：

(1) Tools->Options->Directories 下的 Show directories for 中，include files 下面新增一个 C:\PROGRAM FILES\MICROSOFT SDK\INCLUDE，然后把这个新增的项提到最上面。

(2) Tools->Options->Directories 下的 Show directories for 中，Library files 下面新增一个 C:\PROGRAM FILES\MICROSOFT SDK\LIB，然后把这个新增的项提到最上面。

使用微软的 CAPI 进行编程时，先要知道一些概念。

- 容器：在 CAPI 的接口中有容器的概念，数字证书（证书，公钥和私钥）都是放在容器中的，每个容器有不同的名称，一个容器中只能存放一张签名证书和一张加密证书。如不指定容器名称，ET199Auto 会使用一个默认的 GUID 作为容器名。
- 签名证书：只用来进行签名操作的证书，不能进行加解密操作。RSA 密钥类型为 AT_SIGNATURE。
- 加密证书：可以进行签名和加解密操作。RSA 密钥类型为 AT_KEYEXCHANGE。

在工程中加入 wincrypt.h 头文件(#include <wincrypt.h>)，并加入 Crypt32.lib 的库。

另外 CAPI 接口不能完成所有 PKCS#11 接口所完成的功能，如：修改 PIN 码，读写数据等，您的应用中要用到这些功能就必须使用 PKCS#11 接口来实现。

4.1 枚举 ET199Auto 硬件中的证书

存储在 ET199Auto 中的证书都是使用 EnterSafe 提供的中间件产生或者导入的，ET199Auto 的 CSP 名称为：“EnterSafe ET199Auto CSP V1.0”。枚举 ET199Auto 硬件中的证书代码如下：

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <wincrypt.h>

#define ET199Auto_CSP_NAME "EnterSafe ET199Auto CSP v1.0"

int main(int argc, char* argv[])
{
```

```

//CSP 句柄
HCRYPTPROV hTokenProv = NULL;
//密钥句柄
HCRYPTKEY hKeyCAPI = NULL;

//存放容器名称的数组，容器名最大只能 260 字节
BYTE pbCertContainerName[9][260] = {0};
DWORD dwCertNumber = 0;
DWORD dwCertContainerNameLen = 260;

//证书数据
DWORD dwCertLen = 0;
BYTE* pbCert = NULL;
PCCERT_CONTEXT pCertContext = NULL;

//证书名称
DWORD dwCertNameLen = 0;
BYTE* pbCertName = NULL;

//证书序列号
DWORD dwCertSNLen = 0;
BYTE* pbCertSN = NULL;

DWORD i=0, j=0;

//获得 CSP 句柄。第二个参数给 NULL，先枚举容器
if(!CryptAcquireContext(&hTokenProv, NULL,
    ET199Auto_CSP_NAME, PROV_RSA_FULL, NULL))
{
    printf("CryptAcquireContext Error:0x%08x\n", GetLastError());
    return -1;
}

//枚举第一个容器，最后一个参数 CRYPT_FIRST
if(CryptGetProvParam(
    hTokenProv,
    PP_ENUMCONTAINERS,
    pbCertContainerName[dwCertNumber],
    &dwCertContainerNameLen,
    CRYPT_FIRST))
{
    //打印枚举到的第一个容器名
    printf("Container %d Name: %s\n",
        dwCertNumber, pbCertContainerName[dwCertNumber]);
}

```

```

        if(dwCertContainerNameLen != 0)
            dwCertNumber ++;

//枚举其它的容器，最后一个参数为 0
while(CryptGetProvParam(
        hTokenProv,
        PP_ENUMCONTAINERS,
        pbCertContainerName[dwCertNumber],
        &dwCertContainerNameLen,
        0))
{
    //打印枚举到的容器
    printf("Container %d Name: %s\n",
        dwCertNumber, pbCertContainerName[dwCertNumber]);

    if(dwCertContainerNameLen != 0)
        dwCertNumber ++;

}
}

//打印 ET199Auto 中有几个容器
printf("Certificate Number In ET199Auto: %d\n", dwCertNumber);

//释放 CSP 句柄
CryptReleaseContext(hTokenProv, 0);

if(dwCertNumber == 0)
{
    printf("No Certificate In ET199Auto!\n");
}

//循环获得 ET199Auto 中每个证书的信息
for(i=0; i<dwCertNumber; ++i)
{
    //按容器获得 CSP 句柄。第二个参数给容器名
    if(!CryptAcquireContext(&hTokenProv,
        (const char*)pbCertContainerName[i],
        ET199Auto_CSP_NAME, PROV_RSA_FULL, NULL))
    {
        printf("CryptAcquireContext Error:0x%08x\n", GetLastError());
        return -1;
    }
}

```

```

//获得加密密钥句柄。第二个参数给 AT_KEYEXCHANGE
//如果一个容器中既有加密密钥也有签名密钥，请自行处理
if(!CryptGetUserKey(hTokenProv, AT_KEYEXCHANGE, &hKeyCAPI))
{
    printf("CryptGetUserKey Error:0x%08x\n", GetLastError());
    return -1;
}

//第一次调用获取证书数据长度
if(!CryptGetKeyParam(hKeyCAPI, KP_CERTIFICATE, NULL, &dwCertLen, 0))
{
    printf("CryptGetKeyParam 1 Error:0x%08x\n", GetLastError());
    return -1;
}

//分配空间
pbCert = new BYTE[dwCertLen];
memset(pbCert, 0, dwCertLen);

//第二次调用获取证书数据
if(!CryptGetKeyParam(hKeyCAPI, KP_CERTIFICATE,
    pbCert, &dwCertLen, 0))
{
    printf("CryptGetKeyParam 2 Error:0x%08x\n", GetLastError());
    return -1;
}

//根据证书数据创建 CERT_CONTEXT 结构
pCertContext = CertCreateCertificateContext(
    PKCS_7_ASN_ENCODING | X509_ASN_ENCODING,
    pbCert,
    dwCertLen);

if(pCertContext == NULL)
{
    printf("CertCreateCertificateContext Error:0x%08x\n",
        GetLastError());
    return -1;
}

//获得证书名称
dwCertNameLen = CertGetNameString(pCertContext,
    CERT_NAME_SIMPLE_DISPLAY_TYPE, 0, NULL, NULL, 0);

```

```

pbCertName = new BYTE[dwCertNameLen+1];
memset(pbCertName, 0, dwCertNameLen+1);

CertGetNameString(pCertContext, CERT_NAME_SIMPLE_DISPLAY_TYPE,
    0, NULL, (char *)pbCertName, dwCertNameLen);

printf("\nCert %d Name: %s\n", i, pbCertName);

//获得证书 SN
dwCertSNLen = pCertContext->pCertInfo->SerialNumber.cbData;
pbCertSN = new BYTE[dwCertSNLen];
memset(pbCertSN, 0, dwCertSNLen);
memcpy(pbCertSN, pCertContext->pCertInfo->SerialNumber.pbData,
    dwCertSNLen);

printf("Cert %d SN:\n", i);
for(j=0; j<dwCertSNLen; ++j)
{
    printf("%02x ", pbCertSN[j]);
}
printf("\n");

//释放 CSP 句柄
CryptReleaseContext(hTokenProv, 0);

//释放内存
if(pbCert != NULL)
{
    delete [] pbCert;
    pbCert = NULL;
}

if(pbCertName != NULL)
{
    delete [] pbCertName;
    pbCertName = NULL;
}

if(pbCertSN != NULL)
{
    delete [] pbCertSN;
    pbCertSN = NULL;
}

```

```

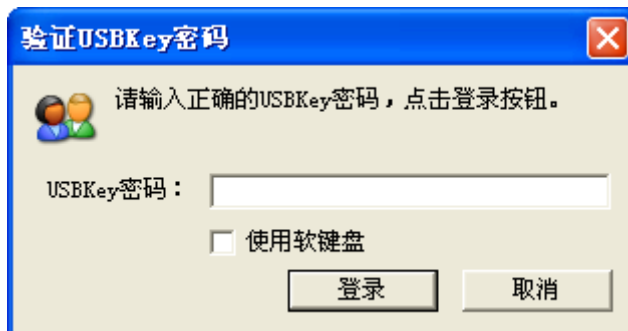
    }

    return 0;
}

```

4.2 使用 CAPI 接口验证用户 PIN (User PIN)

在使用 CAPI 的接口进行私钥签名或者私钥解密这些涉及到私钥的操作时，需要先验证用户 PIN (User PIN)，这时 ET199Auto 会弹出一个输入用户 PIN 码的对话框，如下图所示：



如果开发商不希望弹出对话框，可以使用 CAPI 接口进行用户 PIN 的验证。**用户 PIN 的验证状态是按进程来划分的，当要在同一个进程中恢复为没有验证过的状态，也需要使用下面的方法。**代码如下：

```

HCRYPTPROV hTokenProv = NULL;
BYTE pbUserPIN[] = "12341";

//获得 CSP 句柄
if(!CryptAcquireContext(&hTokenProv, NULL,
    ET199Auto_CSP_NAME, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT))
{
    printf("CryptAcquireContext Error:0x%08x\n", GetLastError());
    return -1;
}

//设置成没有验证过的状态，第三个参数给 NULL
if(!CryptSetProvParam(hTokenProv, PP_SIGNATURE_PIN, NULL, 0))
{
    printf("CryptSetProvParam Error:0x%08x\n", GetLastError());
    return -1;
}

//验证用户 PIN，第三个参数给用户 PIN
if(!CryptSetProvParam(hTokenProv, PP_SIGNATURE_PIN, pbUserPIN, 0))
{

```

```

        printf("CryptSetProvParam Error:0x%08x\n", GetLastError());
        return -1;
    }

    //释放 CSP 句柄
    CryptReleaseContext(hTokenProv, 0);

```

4.3 签名和验签

在进行签名和验签时，要保证 ET199Auto 中至少有一对 RSA 密钥对。代码如下：

```

#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <wincrypt.h>

//定义 CSP 名称
#define TEST_CSP_NAME    "EnterSafe ET199Auto CSP v1.0"
//容器名称
#define CONTAINER_NAME   "CSPKeyTest"

//错误处理函数
void MyHandleError(char *s)
{
    fprintf(stderr, "An error occurred in running the program. \n");
    fprintf(stderr, "%s\n", s);
    fprintf(stderr, "Error number %x.\n", GetLastError());
    fprintf(stderr, "Program terminating. \n");
    exit(1);
}

void main(void)
{
    HCRYPTPROV hProv;                //CSP 句柄
    BYTE *pbBuffer= (BYTE *)"123456"; //被签名的数据
    DWORD dwBufferLen = strlen((char *)pbBuffer); //被签名的数据长度
    HCRYPTHASH hHash;                //HASH 对象句柄
    HCRYPTKEY hKey;                    //密钥句柄
    BYTE *pbKeyBlob;                  //公钥数据，BLOB 结构
    BYTE *pbSignature;                //数字签名
    DWORD dwSigLen;                    //签名长度
    DWORD dwBlobLen;                  //公钥长度，BLOB 结构的长度
    LPTSTR szDescription = "";        //签名时的 Description 参数

```

```

        // 获得 CSP 句柄
printf("The following phase of this program is signature.\n\n");
if(CryptAcquireContext(
    &hProv,
    NULL,
    TEST_CSP_NAME,
    PROV_RSA_FULL,
    0))
{
    printf("CSP context acquired.\n");
}
else    //如获取失败，则密钥容器不存在，创建一个新的容器
{
    if(CryptAcquireContext(
        &hProv,
        NULL,
        TEST_CSP_NAME,
        PROV_RSA_FULL,
        CRYPT_NEWKEYSET))    //创建新的密钥容器
    {
        printf("A new key container has been created.\n");
    }
    else
    {
        MyHandleError("Error during CryptAcquireContext.");
    }
}

// 从密钥容器中取数字签名用的密钥
if(CryptGetUserKey(
    hProv,
    AT_SIGNATURE,    //这里也可以使用 AT_KEYEXCHANGE
    &hKey))
{
    printf("The signature key has been acquired. \n");
}
else//如果取密钥失败，则创建新的密钥，
{
    if(GetLastError() == NTE_BAD_KEYSET)
    {
        if(CryptGenKey(
            hProv,        //CSP 句柄
            AT_SIGNATURE,    //创建的密钥对类型为 AT_SIGNATURE

```



```

        0,                //key 类型，这里用默认值
        &hKey))           //创建成功返回新创建的密钥对的句柄
    {
        printf("Created a signature key pair.\n");
    }
    else
    {
        MyHandleError("Error occurred creating a
                        signature key.\n");
    }
}

else
{
    MyHandleError("Error during CryptGetUserKey for signkey.");
}
}

```

//这里导出公钥，在验证签名时使用。程序的后半部为验签的过程
 //应用时需要将原文，签名结果和公钥发给对方进行验签

//**ET199Auto 的 CSP 只能导出公钥，不能导出私钥**

```

if(CryptExportKey(
    hKey,                //密钥句柄
    NULL,
    PUBLICKEYBLOB,       //导出公钥
    0,
    NULL,
    &dwBlobLen))         //得到公钥的大小
{
    printf("Size of the BLOB for the public key determined. \n");
}
else
{
    MyHandleError("Error computing BLOB length.");
}

```

// 为存储公钥的缓冲区分配内存。

```

if(pbKeyBlob = (BYTE*)malloc(dwBlobLen))
{
    printf("Memory has been allocated for the BLOB. \n");
}
else
{
    MyHandleError("Out of memory. \n");
}

```

```

}

// 真正导出公钥数据
if(CryptExportKey(
    hKey,
    NULL,
    PUBLICKEYBLOB,
    0,
    pbKeyBlob,    //公钥数据，为 BLOB 结构
    &dwBlobLen))
{
    printf("Contents have been written to the BLOB. \n");
}
else
{
    MyHandleError("Error during CryptExportKey.");
}

//在签名前先要进行 HASH 散列操作。创建 HASH 对象
if(CryptCreateHash(
    hProv,        //CSP 句柄
    CALG_SHA1,    //使用 SHA1 散列算法，也可以使用 CALG_MD5,
    0,
    0,
    &hHash))      //HASH 对象句柄
{
    printf("Hash object created. \n");
}
else
{
    MyHandleError("Error during CryptCreateHash.");
}

//对被签名数据进行 HASH 运算
if(CryptHashData(
    hHash,        //HASH 对象句柄
    pbBuffer,     //被签名数据 ( "123456" )
    dwBufferLen,  //被签名数据长度
    0))
{
    printf("The data buffer has been hashed.\n");
}
else
{

```

```

        MyHandleError("Error during CryptHashData.");
    }

//使用私钥对经过 HASH 后的数据进行签名
//这时会弹出输入用户 PIN (User PIN) 的对话框
dwSigLen= 0;
if(CryptSignHash(
    hHash,                //HASH 对象句柄
    AT_SIGNATURE,         //密钥类型，可以是 AT_KEYEXCHANGE
    szDescription,        //这个参数没用
    0,
    NULL,
    &dwSigLen))           //得到数字签名大小
{
    printf("Signature length %d found.\n", dwSigLen);
}
else
{
    MyHandleError("Error during CryptSignHash.");
}

//为数字签名缓冲区分配内存
if(pbSignature = (BYTE *)malloc(dwSigLen))
{
    printf("Memory allocated for the signature.\n");
}
else
{
    MyHandleError("Out of memory.");
}

//得到数字签名
if(CryptSignHash(
    hHash,                //HASH 对象句柄，里面包含被签名数据经过 HASH 后的数据
    AT_SIGNATURE,         //密钥类型，可以是 AT_KEYEXCHANGE
    szDescription,
    0,
    pbSignature,          //这里将返回数字签名
    &dwSigLen))
{
    printf("pbSignature is the hash signature.\n");
}
else
{

```

```

        MyHandleError("Error during CryptSignHash.");
    }

    //打印签名结果
    printf("\n");
    for(DWORD i=0; i<dwSigLen; ++i)
    {
        if(i%16 == 0)
            printf("\n");
        printf("%02x ",pbSignature[i]);
    }
    printf("\n");

    //释放 HASH 对象句柄
    if(hHash)
        CryptDestroyHash(hHash);

    printf("The hash object has been destroyed.\n");
    printf("The signing phase of this program is completed.\n\n");

```

/*****

上面的代码为签名过程。 下面的代码是接收者使用的,这里为了说明方便就把它放在一个文件里了。pbBuffer, pbSignature, szDescription, pbKeyBlob, 还有他们的长度在这里直接使用了, 应该是接收者从文件或其他地方读取的。pbBuffer 里保存的是被签名的数据, 所以认证时的内容必须跟签名时的一样这里使用的 CSP 句柄也没有重新创建

*****/

```

    printf("The following phase of this program is verify signature.\n\n");

    //使用公钥进行验签。先把公钥数据导入生成公钥句柄
    HCRYPTKEY hPubKey;
    if(CryptImportKey(
        hProv,          //CSP 句柄
        pbKeyBlob,      //公钥数据, BLOB 结构
        dwBlobLen,      //公钥长度
        0,
        0,
        &hPubKey))  //公钥句柄
    {
        printf("The key has been imported.\n");
    }

```

```

else
{
    MyHandleError("Public key import failed.");
}

//先对原文做散列运算，验签时使用。创建哈希对象
if(CryptCreateHash(
    hProv,          //CSP 句柄
    CALG_SHA1,     //这里使用 SHA1 散列算法，与签名时的一致
    0,
    0,
    &hHash))       //HASH 对象句柄
{
    printf("The hash object has been recreated. \n");
}
else
{
    MyHandleError("Error during CryptCreateHash.");
}

//将原文做 HASH 计算
if(CryptHashData(
    hHash,          //HASH 对象句柄
    pbBuffer,       //原文 ( "123456" )
    dwBufferLen,    //原文长度
    0))
{
    printf("The new hash has been created. \n");
}
else
{
    MyHandleError("Error during CryptHashData.");
}

//验证数字签名，即原文 HASH 后的数据，签名后的数据和公钥进行验证操作
if(CryptVerifySignature(
    hHash,          //HASH 对象句柄，里面包含原文 HASH 后的数据
    pbSignature,    //签名后的数据
    dwSigLen,       //签名后数据的程度
    hPubKey,        //公钥句柄
    szDescription,  //没有用
    0))
{
    printf("The signature has been verified. \n");
}

```

```

    }
    else
    {
        printf("Signature not validated!\n");
    }

    //释放内存
    if(pbSignature)
        free(pbSignature);

    if(pbKeyBlob)
        free(pbKeyBlob);

    //释放 HASH 对象句柄
    if(hHash)
        CryptDestroyHash(hHash);

    //释放 CSP 句柄
    if(hProv)
        CryptReleaseContext(hProv, 0);

    printf("\n\nPress any key to exit...\n");
    getch();
}

```

4.4 RSA 加解密

```

HCRYPTPROV hTokenProv = NULL;    //CSP 句柄
HCRYPTKEY hKeyCAPI = NULL;      //密钥句柄

BYTE pbData[] = "Hello World!"; //待加密数据
DWORD dwDataLen = sizeof(pbData); //待加密数据长度（为 13）

BYTE* pbEncryptedData = NULL;    //加密后数据
DWORD dwEncryptedLen = dwDataLen;
DWORD dwTemp = 0;

//获得 CSP 句柄
if(!CryptAcquireContext(&hTokenProv, NULL,
    ET199Auto_CSP_NAME, PROV_RSA_FULL, NULL))
{
    printf("CryptAcquireContext Error:0x%08x\n", GetLastError());
}

```

```

        return -1;
    }

    //获得加密密钥句柄
    if(!CryptGetUserKey(hTokenProv, AT_KEYEXCHANGE, &hKeyCAPI))
    {
        printf("CryptGetUserKey Error:0x%08x\n", GetLastError());
        return -1;
    }

    //第一次调用，获得加密后的数据长度
    if(!CryptEncrypt(
        hKeyCAPI,          //密钥句柄
        0,
        TRUE,              //当待加密数据只有一块或者是最后一块时设为 TRUE
        0,
        NULL,              //输入为待加密数据，返回为密文。第一次调用给 NULL
        &dwEncryptedLen,    //输入为待加密数据的长度，返回为加密后数据的长度
        dwDataLen))        //缓冲区大小
    {
        printf("CryptEncrypt 1 Error:0x%08x\n", GetLastError());
        return -1;
    }

    //分配空间
    pbEncryptedData = new BYTE[dwEncryptedLen];
    memset(pbEncryptedData, 0, dwEncryptedLen);

    //将待加密的数据拷贝到分配的空间中
    memcpy(pbEncryptedData, pbData, dwDataLen);

    //设置第六个参数，待加密的数据长度（为 13）
    //设置第七个参数，缓冲区的大小（为 128）
    dwTemp = dwEncryptedLen;
    dwEncryptedLen = dwDataLen;
    dwDataLen = dwTemp;

    //第二次调用，得到加密后的密文
    if(!CryptEncrypt(hKeyCAPI, 0, TRUE, 0,
        pbEncryptedData, &dwEncryptedLen, dwDataLen))
    {
        printf("CryptEncrypt 2 Error:0x%08x\n", GetLastError());
        return -1;
    }

```

```

//打印加密结果
printf("Cipher Text:\n");
for(i=0; i<dwEncryptedLen; ++i)
{
    printf("%02X ", pbEncryptedData[i]);

    if((i+1) % 16 == 0)
        printf("\n");
}

//解密，第六个参数一定要是 128 的倍数（RSA 为 1024 位时）
if(!CryptDecrypt(
    hKeyCAPI,          //密钥句柄
    0,
    TRUE,              //当待加密数据只有一块或者是最后一块时设为 TRUE
    0,
    pbEncryptedData,   //输入为加密后的密文，输出为解密后的原文
    &dwEncryptedLen))//输入为加密后密文长度，输出为解密后原文长度
{
    printf("CryptDecrypt Error:0x%08x\n", GetLastError());
    return -1;
}

//打印原文
printf("\nPlain Text:\n%s\n", pbEncryptedData);

//释放 CSP 句柄
CryptReleaseContext(hTokenProv, 0);

//释放内存
if(pbEncryptedData != NULL)
{
    delete [] pbEncryptedData;
    pbEncryptedData = NULL;
}

```


五、ET199Auto 参数

| 中间件参数 | |
|----------------|---------------------------------|
| 支持的操作系统 | Windows 2000/XP/2003/Vista/2008 |
| X64 系统 | 支持 |
| 远程解锁 | 支持 |
| ISO 空间是否可动态设置 | 可以 |
| CSP 名称 | EnterSafe ET199Auto CSP V1.0 |
| CSP 软键盘 | 支持 |
| 软件语言 | 中英繁自适应 |
| 证书和标准 | MS CAPI, X.509 v3 证书存储, SSL v3 |
| AutoRun | 支持 |
| AutoRun 是否有界面 | Mini 界面 |
| ET199Auto 硬件参数 | |
| 主芯片类型 | 高性能国外进口 16 位智能卡芯片 |
| 用户空间 | 16K |
| RSA 密钥对支持 | 512/1024/2048 |
| 通讯加密 | 全部加密 |
| 设备类型 | SCSI-CDROM |
| 内置安全算法 | RSA, DES, 3DES, MD5, SHA-1 |
| 芯片安全水平 | 安全加密的数据存储 |

| | |
|--------|------------------|
| 功率 | < 250 mW |
| 工作温度 | 0 ~ 70° C |
| 存放温度 | - 40 ~ 85° C |
| 湿度 | 0 ~ 100%不结露 |
| 接口类型 | A 类 USB |
| 外壳 | 一次性，防水，硬塑料外壳 |
| 数据存储年限 | 室温下数据保持时间最少 10 年 |
| 写次数 | 最少擦写次数 10 万次 |