CryptoAPI 培训教程

项目名称：无	开发部门：无
项目编号：无	密　　级：无
作　　者：Ady Lee	版　　本：1.2
编写日期：2002-3-1	审　　核：无
电子邮件：adylee@sina.com	

更新记录

修改人	修改日期	修改对象	备注（原因、进一步的说明等）
Ady Lee	2002-3-1	整篇	• 丰富内容
Ady Lee	2002-3-11	整篇	• 丰富内容
Ady Lee	2003-9-29	整篇	• 整理

目录

1 前言.....3

2 目的.....3

3 新功能.....3

4 CRYPTOAPI.....3

4.1 基本加密函数.....3

4.1.1 服务提供者函数.....4

4.1.2 密钥的产生和交换函数.....4

4.1.3 编码/解码函数.....5

4.1.4 数据加密/解密函数.....5

4.1.5 哈希和数字签名函数.....5

4.1.6 函数详解.....6

4.2 证书和证书库函数.....24

4.2.1 证书库函数.....24

4.2.2 维护函数.....24

4.2.3 证书函数.....24

4.2.4 证书撤销列表函数.....25

4.2.5 证书信任列表函数.....26

4.2.6 扩展属性函数.....26

4.2.7 函数详解.....26

4.3 证书验证函数.....27

4.3.1 使用 CTL 的函数.....28

4.3.2 证书链验证函数.....28

4.4 消息函数.....28

4.4.1 低级消息函数.....28

4.4.2 简化消息函数.....29

4.5 辅助函数.....29

4.5.1 数据管理函数.....29

4.5.2 数据转换函数.....30

4.5.3 增强密钥用法函数.....30

4.5.4 密钥标示函数.....31

4.5.5 证书库回调函数.....31

4.5.6 OID 支持函数.....31

4.5.7 远程对象恢复函数.....32

4.5.8 PFX 函数.....32

5 待续.....32

1 前言

事隔 1 年有余，重新整理此文档，感触颇深。冷眼旁观 MS 的 CryptoAPI 飞速发展，如今的 CAPI 已趋于稳定。MS 还在 CryptoAPI 的基础上封装了一个 CAPICOM（目前版本 2.0），以供其他语言使用。

创建此文档是一年前，当时由于工作需要在 Windows 上写一个桌面安全的软件，所以接触到了 CryptoAPI，顺便自己写了一个教程以备以后使用。由于当初对 CAPI 认识有限，文档中难免会有错误和不足，不过以后我会不断的来更新和维护这个文档，以迎合 MS CAPI 的发展，也希望各位朋友能够一如既往的支持我。

由于工作之余匆忙整理，文档中难免有错误和遗漏，欢迎广大读者批评指正。作者信箱为 adylee@sina.com。

2 目的

CryptoAPI（一个应用程序编程接口）目的就是提供开发者在 Windows 下使用 PKI 的编程接口。CryptoAPI 提供了很多函数，包括编码、解码、加密、解密、哈希、数字证书、证书管理和证书存储等功能。对于加密和解密，CryptoAPI 同时提供基于会话密钥和公钥/私钥对的方法。

3 新功能

CryptoAPI 2.0 提供了用于密码学的函数。2.0 版本包括了基本加密函数。这些函数提供了大部分安全通讯的重要方面——秘密性和完整性。

其它重要的方面是认证——要保证发送者和接收者身份的确定性。一种方法就是使用数字证书，有一种趋势就是要把证书合并到安全通讯中。CryptoAPI 2.0 中提供了管理和使用证书的函数，另外，2.0 版本中也提供了编码和解码 PKCS#7（ASN.1）消息。

证书管理函数维护和管理了一个证书的可持续库——证书库。CryptoAPI 2.0 在两个级别上加入了消息管理函数——低级消息函数和简化消息函数。低级消息函数比简化消息函数更加富有弹性，但也需要更多的函数调用。

4 CryptoAPI

4.1 基本加密函数

基本加密函数为开发加密应用程序提供了足够灵活的空间。所有 CSP 的通讯都是通过这些函数。

一个 CSP 是实现所有加密操作的独立模块。在每一个应用程序中至少需要提供一个 CSP 来完成所需的加密操作。

如果使用多于一个以上的 CSP，在加密函数调用中就要指定所需的 CSP。微软基本加密提供者（Microsoft Base Cryptographic Provider），是缺省绑定到 CryptoAPI 里的。如果没有指定其他 CSP 时，这个 CSP 就是却省的。

每一个 CSP 对 CryptoAPI 提供了一套不同的实现。一些提供了更加强大的加密算法，而其他一些 CSP 包含了对硬件的支持，比如智能卡。另外，一些 CSP 偶尔和使用者直接通讯，比如数字签名就使用了用户的签名私钥。

基本加密函数包含了以下几种：

4.1.1 服务提供者函数

应用程序使用服务提供者函数来连接和断开一个 CSP。下面就是主要的 API：

CryptAcquireContext	获得指定 CSP 的密钥容器的句柄
CryptContextAddRef	对 HCRYPTPROV 句柄增加一个应用计数
CryptEnumProviders	枚举当前计算机中的 CSP
CryptEnumProviderTypes	枚举 CSP 的类型
CryptGetDefaultProvider	对于指定 CSP 类型的却省 CSP
CryptGetProvParam	得到一个 CSP 的属性
CryptInstallDefaultContext	安装先前得到的 HCRYPTPROV 上下文作为当前却省的上下文
CryptReleaseContext	释放由 CryptAcquireContext 得到的句柄
CryptSetProvider 和 CryptSetProviderEx	为指定 CSP 类型指定一个却省的 CSP
CryptSetProvParam	指定一个 CSP 的属性
CryptUninstallDefaultContext	删除先前由 CryptInstallDefaultContext 安装的却省上下文

4.1.2 密钥的产生和交换函数

密钥产生函数创建、配置和销毁加密密钥。他们也用于和其他用户进行交换密钥。下面就是主要的一些函数：

CryptAcquireCertificatePrivateKey	对于指定证书上下文得到一个 HCRYPTPROV 句柄和 dwKeySpec
CryptDeriveKey	从一个密码中派生一个密钥
CryptDestoryKey	销毁密钥
CryptDuplicateKey	制作一个密钥和密钥状态的精确复制
CryptExportKey	把 CSP 的密钥做成 BLOB 传送到应用程序的内存空间中
CryptGenKey	创建一个随机密钥
CryptGenRandom	产生一个随机数
CryptGetKeyParam	得到密钥的参数
CryptGetUserKey	得到一个密钥交换或签名密钥的句柄

CryptImportKey	把一个密钥 BLOB 传送到 CSP 中
CryptSetKeyParam	指定一个密钥的参数

4.1.3 编码/解码函数

有一些编码/解码函数，他们可以用来对证书、证书撤销列表、证书请求和证书扩展进行编码和解码。

以下就是这几个函数：

CryptDecodeObject	对 lpszStructType 结构进行解码
CryptDecodeObjectEx	对 lpszStructType 结构进行解码，此函数支持内存分配选项
CryptEncodeObject	对 lpszStructType 结构进行编码
CyptEncodeObjectEx	对 lpszStructType 结构进行编码，此函数支持内存分配选项

4.1.4 数据加密/解密函数

这些函数支持数据的加密/解密操作。CryptEncrypt 和 CryptDecrypt 要求在被调用前指定一个密钥。这个密钥可以由 CryptGenKey、CryptDeriveKey 或 CryptImportKey 产生。创建密钥时要指定加密算法。CryptSetKeyParam 函数可以指定额外的加密参数。

CryptDecrypt	使用指定加密密钥来解密一段密文
CryptEncrypt	使用指定加密密钥来加密一段明文
CryptProtectData	执行对 DATA_BLOB 结构的加密
CryptUnprotectData	执行对 DATA_BLOB 结构的完整性验证和解密

4.1.5 哈希和数字签名函数

这些函数在应用程序中完成计算哈希、创建和校验数字签名。

CryptCreateHash	创建一个空哈希对象
CryptDestoryHash	销毁一个哈希对象
CryptDuplicateHash	复制一个哈希对象
CryptGetHashParam	得到一个哈希对象参数
CryptHashData	对一块数据进行哈希，把它加到指定的哈希对象中
CryptHashSessionKey	对一个会话密钥进行哈希，把它加到指定的哈希对象中
CryptSetHashParam	设置一个哈希对象的参数
CryptSignHash	对一个哈希对象进行签名

CryptVerifySignature	校验一个数字签名

4.1.6 函数详解

4.1.6.1 获得 CSP 密钥容器句柄

4.1.6.1.1 CryptAcquireContext

```
BOOL WINAPI CryptAcquireContext(
  HCRYPTPROV *phProv,
  LPCTSTR pszContainer,
  LPCTSTR pszProvider,
  DWORD dwProvType,
  DWORD dwFlags
);
```

参数：

phProv

[out]CSP 句柄指针

pszContainer

[in]密钥容器名称，指向密钥容器的字符串指针。如果 dwFlags 为 CRYPT_VERIFYCONTEXT，pszContainer 必须为 NULL。

pszProvider

[in]指向 CSP 名称的字符串指针。如果为 NULL，就使用却省的 CSP。

dwProvType

[in]CSP 类型。

dwFlags

[in]标志。

CRYPT_VERIFYCONTEXT

此选项指出应用程序不需要使用公钥/私钥对，如程序只执行哈希和对称加密。只有程序需要创建签名和解密消息时才需要访问私钥。

CRYPT_NEWKEYSET

使用指定的密钥容器名称创建一个新的密钥容器。如果 pszContainer 为 NULL，密钥容器就使用却省的名称创建。

CRYPT_MACHINE_KEYSET

由此标志创建的密钥容器只能由创建者本人或有系统管理员身份的人使用。

CRYPT_DELETEKEYSET

删除由 pszContainer 指定的密钥容器。如果 pszContainer 为 NULL，却省名称的容器就会被删除。此容器里的所有密钥对也会被删除。

CRYPT_SLIENT

应用程序要求 CSP 不显示任何用户界面。

说明：

这个函数是用来取得指定 CSP 密钥容器句柄，以后的任何加密操作就是针对此 CSP 句

柄而言。函数首先查找由 dwProvType 和 pszProvider 指定的 CSP，如果找到了 CSP，函数就查找由此 CSP 指定的密钥容器。由适当的 dwFlags 标志，这个函数就可以创建和销毁密钥容器，如果不要求访问私钥的话，也可以提供对 CSP 临时密钥容器的访问。

4.1.6.1.2 CryptReleaseContext

```
BOOL WINAPI CryptReleaseContext(
  HCRYPTPROV hProv,
  DWORD dwFlags
);
```

参数：

hProv

[in]由 CryptAcquireContext 获得的 CSP 句柄。

dwFlags

[in]保留。必须为 0。

说明：

此函数释放 CSP 的句柄。对于每一次调用，CSP 的引用计数都减 1。当引用计数为 0 时，CSP 上下文就会被系统释放变成无效句柄，以后针对此 CSP 句柄的函数不再可用。此函数并不销毁密钥容器或密钥对。

```
//--------------------------------------------------------------------

HCRYPTPROV hCryptProv;
if (CryptAcquireContext(
    hCryptProv, NULL,
    MS_DEF_PROV,
    PROV_RSA_FULL,
    CRYPT_VERIFYCONTEXT))
    CryptReleaseContext(hCryptProv, NULL);
```

4.1.6.2 枚举 CSP

4.1.6.2.1 CryptEnumProviders

```
BOOL WINAPI CryptEnumProviders(
  DWORD dwIndex,
  DWORD *pdwReserved,
  DWORD dwFlags,
  DWORD *pdwProvType,
  LPTSTR pszProvName,
  DWORD *pcbProvName
);
```

参数：

dwIndex

[in]枚举下一个 CSP 的索引。

pdwReserved

[in]保留。必须为 NULL。

dwFlags

[in]保留。必须为 NULL。

pdwProvType

[out]CSP 的类型。

pszProvName

[out]指向接收 CSP 名称的缓冲区字符串指针。此指针可为 NULL，用来得到字符串的大小。

pcbProvName

[in/out]指出 pszProvName 字符串的大小。

说明：

此函数得到第一个或下一个可用的 CSP。如果使用循环，就可以得到计算机上所有可用的 CSP。

```
//--------------------------------------------------------------------
//声明初始化数据

DWORD           cbName;
DWORD           dwType;
DWORD           dwIndex=0;
CHAR            *pszName;

//循环枚举 CSP.

dwIndex = 0;
while(CryptEnumProviders(
        dwIndex,
        NULL,
        0,
        &dwType,
        NULL,           // 如果为 NULL，则 cbName 返回 CSP 名称所需字节数
        &cbName         //
        ))
{

//--------------------------------------------------------------------
//
//动态分配内存.

if (!(pszName = (LPTSTR)LocalAlloc(LMEM_ZEROINIT, cbName)))
{
    printf("ERROR - LocalAlloc failed!");
```

```
    exit(1);
}
//--------------------------------------------------------------------
// 获得 CSP 名称

if (CryptEnumProviders(
        dwIndex++,
        NULL,
        0,
        &dwType,
        pszName,
        &cbName
        ))
{
    printf ("%4.0d %s\n",dwType, pszName);
}
else
{
    printf("ERROR - CryptEnumProviders");
    exit(1);
}
LocalFree(pszName);

} // 循环结束
```

4.1.6.3 获得 CSP 参数

4.1.6.3.1 CryptGetProvParam

```
BOOL WINAPI CryptGetProvParam(
  HCRYPTPROV hProv,
  DWORD dwParam,
  BYTE *pbData,
  DWORD *pdwDataLen,
  DWORD dwFlags
);
```

参数：

hProv

[in]CSP 句柄。

dwParam

[in]指定查询的参数。

PP_CONTAINER	指向密钥名称的字符串
PP_ENUMALGS	不断的读出 CSP 支持的所有算法
PP_ENUMALGS_EX	比 PP_ENUMALGS 获得更多的算法信息
PP_ENUMCONTAINERS	不断的读出 CSP 支持的密钥容器

PP_IMPTYPE 指出 CSP 怎样实现的
PP_NAME 指向 CSP 名称的字符串
PP_VERSION CSP 的版本号
PP_KEYSIZE_INC AT_SIGNATURE 的位数
PP_KEYX_KEYSIZE_INC AT_KEYEXCHANGE 的位数
PP_KEYSET_SEC_DESCR 密钥的安全描述符
PP_UNIQUE_CONTAINER 当前密钥容器的唯一名称
PP_PROVTYPE CSP 类型
PP_USE_HARDWARE_RNG 指出硬件是否支持随机数发生器
PP_KEYSPEC 返回 CSP 密钥的信息

pbData
[out]指向接收数据的缓冲区指针。
pdwDataLen
[in/out]指出 pbData 数据长度。
dwFlags
[in]如果指定 PP_ENUMCONTAINERS，就指定 CRYPT_MACHINE_KEYSET。

说明：
此函数获得 CSP 的各种参数。

```
//--------------------------------------------------------------------
//

HCRYPTPROV hCryptProv;
BYTE       pbData[1000];

DWORD cbData;

//--------------------------------------------------------------------
//却省 CSP 名称

cbData = 1000;
if(CryptGetProvParam(
    hCryptProv,
    PP_NAME,
    pbData,
    &cbData,
    0))
{
    printf("CryptGetProvParam succeeded.\n");
    printf("Provider name: %s\n", pbData);
}
else
{
    printf("Error reading CSP name. \n");
    exit(1);
}
```

```
//--------------------------------------------------------------------


cbData = 1000;
if(CryptGetProvParam(
      hCryptProv,
      PP_CONTAINER,
      pbData,
      &cbData,
      0))
{
      printf("CryptGetProvParam succeeded. \n");
      printf("Key Container name: %s\n", pbData);
}
else
{
      printf("Error reading key container name. \n");
      exit(1);
}
```

4.1.6.4 创建哈希

4.1.6.4.1 CryptCreateHash

```
BOOL WINAPI CryptCreateHash(
  HCRYPTPROV hProv,
  ALG_ID Algid,
  HCRYPTKEY hKey,
  DWORD dwFlags,
  HCRYPTHASH *phHash
);
```

参数：

hProv

[in]CSP 句柄

Algid

[in]哈希算法的标示符。

hKey

[in]如果哈希算法是密钥哈希，如 HMAC 或 MAC 算法，就用此密钥句柄传递密钥。对于非密钥算法，此参数为 NULL。

dwFlags

[in]保留。必须为 0。

phHash

[out]哈希对象的句柄。

说明：

此函数初始化哈希数据流。它创建并返回了一个 CSP 哈希对象的句柄。此句柄由

CryptHashData 和 CryptHashSessionKey 来调用。

4.1.6.4.2 CryptHashData

```
BOOL WINAPI CryptHashData(
  HCRYPTHASH hHash,
  BYTE *pbData,
  DWORD dwDataLen,
  DWORD dwFlags
);
```

参数：

hHash

[in]哈希对象句柄

pbData

[in]指向要加入到哈希对象的数据指针

dwDataLen

[in]数据长度

dwFlags

[in]标志

CRYPT_USERDATA　　所有微软 CSP 都忽略此参数。所有其他 CSP 都不能忽略此参数，如果置此参数，CSP 提示用户直接数据数据。

说明：

此函数把一段数据加入到指定的哈希对象中去。

4.1.6.4.3 CryptGetHashParam

```
BOOL WINAPI CryptGetHashParam(
  HCRYPTHASH hHash,
  DWORD dwParam,
  BYTE *pbData,
  DWORD *pdwDataLen,
  DWORD dwFlags
);
```

参数：

hHash

[in]哈希对象的句柄

dwParam

[in]查询类型。可以是下列：

HP_ALGID　　哈希算法

HP_HASHSIZE　哈希值长度

HP_HASHVAL　哈希值，由 hHash 指定的哈希值或者消息哈希

说明：

此函数得到指定哈希对象的数据。

4.1.6.4.4 CryptDestroyHash

```
BOOL WINAPI CryptDestroyHash(
  HCRYPTHASH hHash
);
```

参数：

hHash

[in]要销毁的哈希对象句柄

说明：

此函数销毁由 hHash 指定的哈希对象。当一个哈希对象被销毁后，它对程序来说不可用。

```
…
HCRYPTHASH hCryptHash;
if (CryptCreateHash(
      hCryptProv,
      CALG_MD5,
      0,
      0,
      &hCryptHash
      ))
    CryptDestroyHash(hCryptHash);
…
```

4.1.6.5 派生密钥

4.1.6.5.1 CryptDeriveKey

```
BOOL WINAPI CryptDeriveKey(
  HCRYPTPROV hProv,
  ALG_ID Algid,
  HCRYPTHASH hBaseData,
  DWORD dwFlags,
  HCRYPTKEY *phKey
);
```

参数：

hProv

[in]CSP 句柄

Algid

[in]要产生密钥的对称加密算法

hBaseData

[in]哈希对象的句柄

dwFlags

[in]指定密钥的类型

CRYPT_CREATE_SALT　　典型地，由哈希值产生一个会话密钥，有一些需要补位。如果用此标志，密钥将会赋予一个盐 值

CRYPT_EXPORTABLE	如果置此标志，密钥就可以用 CryptExportKey 函数导出。
CRYPT_NO_SALT	如果置此标志，表示 40 位的密钥不需要分配盐值。
CRYPT_UPDATE_KEY	有些 CSP 从多个哈希值中派生会话密钥。如果这种情况，CryptDeriveKey 需要多次调用。

phKey

[in/out]密钥的句柄

说明：

此函数从一基本数据值中派生会话密钥。函数保证当 CSP 和算法相同时，从相同基本数据值中产生的密钥是唯一的。

4.1.6.5.2 CryptDestroyKey

```
BOOL WINAPI CryptDestroyKey(
  HCRYPTKEY hKey
);
```

参数：

hKey

[in]需要销毁的密钥句柄

说明：

此函数释放密钥句柄。

```
…
HCRYPTKEY hCryptKey;
if (CryptDeriveKey(hCryptProv, m_algid, m_hHash, 0, &hCryptKey))
    CryptDestroyKey(hCryptKey);
…
```

4.1.6.6 加密/解密

4.1.6.6.1 CryptEncrypt

```
BOOL WINAPI CryptEncrypt(
  HCRYPTKEY hKey,
  HCRYPTHASH hHash,
  BOOL Final,
  DWORD dwFlags,
  BYTE *pbData,
  DWORD *pdwDataLen,
  DWORD dwBufLen
);
```

参数：

hKey

[in]加密密钥的句柄

hHash

[in]哈希对象的句柄。如果数据需要同时被哈希并且加密，hHash 就指出了哈希对象。

Final

[in]指出是否是最后一次加密操作。如果 Final 为 TRUE，就为最后一次，否则为 FALSE。

dwFlags

[in]保留

pbData

[in/out]指向被加密的数据地址。

pdwDataLen

[in/out]指向一个 DWORD 值的地址。在调用此函数前，这个值就是需要加密的数据长度。在调用此函数后，这个值就是已经加密的数据长度。如果此值为 NULL，函数就返回需要数据的长度。

dwBufferLen

[in]指出 pbData 的数据长度。

说明：

此函数用于加密数据。加密数据所需要的算法由 hKey 的密钥指定。

4.1.6.6.2 CryptDecrypt

```
BOOL WINAPI CryptDecrypt(
  HCRYPTKEY hKey,
  HCRYPTHASH hHash,
  BOOL Final,
  DWORD dwFlags,
  BYTE *pbData,
  DWORD *pdwDataLen
);
```

参数：

hKey

[in]解密密钥的句柄

hHash

[in]哈希对象的句柄。如果需要解密数据并且同时作哈希，hHash 传递此参数。

Final

[in]指出是否是最后一次解密操作。

dwFlags

[in]保留

pbData

[in/out]需要解密数据的地址

pdwDataLen

[in/out]指向 DWORD 值的指针，此值指出解密数据的长度。在调用此函数前，此值为需要解密数据的长度，调用此函数后，此值为已经解密的数据长度。

说明：

此函数对由 CryptEncrypt 加密过的数据进行解密。

```
//--------------------------------------------------
HCRYPTPROV hCryptProv;
HCRYPTHASH hCryptHash;
HCRYPTKEY hCryptKey;

CryptAcquireContext(
      hCryptProv, NULL,
      MS_DEF_PROV,
      PROV_RSA_FULL,
      CRYPT_VERIFYCONTEXT)

CryptCreateHash(
      hCryptProv,
      CALG_MD5,
      0,
      0,
      &hCryptHash
      )

static char szHash[]=”PISAHASHDATA”; //原始字符串
DWORD dwLen=strlen(szHash);
CryptHashData(hCryptHash, (BYTE*)szHash, dwLen, 0);
CryptDeriveKey(hCryptProv, CALG_RC2, hCryptHash, 0, &hCryptKey);

static char szEntry[]= “PISA2002”;
DWORD dwLenIn = strlen(szEntry);
DWORD dwLenOut=dwLenIn;
CryptEncrypt(hCryptKey, 0, TRUE, 0, (BYTE*)szEntry, &dwLenOut, dwLenIn);
CryptDecrypt(hCryptKey, 0, TRUE, 0,(BYTE*)szEntry,&dwLenOut);

CryptDestroyKey(hCryptKey);
CryptDestroyHash(hCryptHash);
CryptReleaseContext(hCryptProv, NULL);
```

4.1.6.7 签名/验证

4.1.6.7.1 CryptSignMessage

```
BOOL WINAPI CryptSignMessage(
   PCRYPT_SIGN_MESSAGE_PARA pSignPara,
   BOOL fDetachedSignature,
   DWORD cToBeSigned,
   const BYTE *rgpbToBeSigned[ ],
   DWORD rgcbToBeSigned[ ],
```

```
    BYTE *pbSignedBlob,
    DWORD *pcbSignedBlob
);
```

参数：

pSignPara

[in]指向 CRYPT_SIGN_MESSAGE_PARA 结构的指针。

CRYPT_SIGN_MESSAGE_PARA 结构如下：

```
typedef struct _CRYPT_SIGN_MESSAGE_PARA {
    DWORD                           cbSize;
    DWORD                           dwMsgEncodingType;
    PCCERT_CONTEXT                  pSigningCert;
    CRYPT_ALGORITHM_IDENTIFIER HashAlgorithm;
    void                            *pvHashAuxInfo;
    DWORD                           cMsgCert;
    PCCERT_CONTEXT                  *rgpMsgCert;
    DWORD                           cMsgCrl;
    PCCRL_CONTEXT                   *rgpMsgCrl;
    DWORD                           cAuthAttr;
    PCRYPT_ATTRIBUTE                rgAuthAttr;
    DWORD                           cUnauthAttr;
    PCRYPT_ATTRIBUTE                rgUnauthAttr;
    DWORD                           dwFlags;
    DWORD                           dwInnerContentType;
#ifdef CRYPT_SIGN_MESSAGE_PARA_HAS_CMS_FIELDS
    CRYPT_ALGORITHM_IDNETIFIER HashEncryptionAlgorithm;
    void                             pvHashEncryptionAuxInfo;
#endif
} CRYPT_SIGN_MESSAGE_PARA, *PCRYPT_SIGN_MESSAGE_PARA;
```

cbSize

此结构的大小。

dwMsgEncodingType

使用的编码类型。一般为 X509_ASN_ENCODING | PKCS_7_ASN_ENCODING

pSigningCert

指向要签名的 CERT_CONTEXT 指针。

HashAlgorithm

CRYPT_ALGORITHM_IDENTIFIER 指出了对要进行签名的数据进行哈希的哈希算法

pvHashAuxInfo

必须为 NULL

cMsgCert

rgpMsgCert 数组中 CERT_CONTEXT 结构的元素数量。如果此值为 0，则签名消息中不包含任何证书。

rgpMsgCert

指向 CERT_CONTEXT 的数组指针。如果包含 pSigningCert，它的指针必须放到 rgpMsgCert 数组中。

cMsgCrl

rgpMsgCrl 数组指向 CRL_CONTEXT 结构的元素数量。如果为 0，签名消息中不包含任何 CRL_CONTEXT 结构。

rgpMsgCrl

指向 CRL_CONTEXT 结构的数组指针。

cAuthAttr

必须为 0

rgAuthAttr

指向 CRYPT_ATTRIBUTE 数组的指针。每一次都包含了认证信息。

cUnauthAttr

rgUnauthAttr 数组大小。

rgUnauthAttr

指向 CRYPT_ATTRIBUTE 结构的数组指针。

dwFlags

通常为 0。

dwInnerContentType

通常为 0。

HashEncryptionAlgorithm

CRYPT_ALGORITHM_IDENTIFIER 结构。通常为 0

pvHashEncryptionAuxInfo

必须为 0。

fDetachedSignature

[in]如果为 TRUE，就是已解邦定的签名，否则为 FALSE。如果此参数为 TRUE，pbSignedBlob 中只有签名哈希。否则 rgpbToBeSigned 和签名哈希都要被编码。

cToBeSigned

[in]指出 rgpbToBeSigned 数据元素的个数。除非 fDetachedSigned 为 TRUE，此参数就是 1。

rgpbToBeSigned

[in]指向要签名数据的数组指针。

pbSignedBlob

[out]指向一个接收签名哈希的数据地址。如果此参数为 NULL，就是需要来接收数据的内存大小。

pcbSignedBlob

[in/out]指向 DWORD 的地址，此数据指出 pbSignedBlob 的大小。

说明：

此函数对指定数据进行哈希，然后对哈希值进行签名，然后对原始消息和签名哈希进行编码。

```
//--------------------------------------------------------------------
…
#define MY_TYPE    (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
#define SIGNER_NAME    L"Insert_signer_name_here"
#define CERT_STORE_NAME    L"MY"

void HandleError(char *s);
```

```
void main(void)
{
// 系统证书库句柄

HCERTSTORE hStoreHandle;

//--------------------------------------------------------------------
// 待签名的消息

BYTE* pbMessage =
      (BYTE*)"CryptoAPI is a good way to handle security";

//--------------------------------------------------------------------


DWORD cbMessage = strlen((char*) pbMessage)+1;

//--------------------------------------------------------------------
// 证书的上下文

PCCERT_CONTEXT pSignerCert;

CRYPT_SIGN_MESSAGE_PARA    SigParams;
DWORD cbSignedMessageBlob;
BYTE    *pbSignedMessageBlob;
DWORD cbDecodedMessageBlob;
BYTE    *pbDecodedMessageBlob;
CRYPT_VERIFY_MESSAGE_PARA VerifyParams;

//--------------------------------------------------------------------


const BYTE* MessageArray[] = {pbMessage};
DWORD MessageSizeArray[1];
MessageSizeArray[0] = cbMessage;

//--------------------------------------------------------------------
//

printf("Begin processing. \n");

printf(" The message to be signed is\n-> %s.\n",pbMessage);

//--------------------------------------------------------------------
```

```
// Open a certificate store.

if ( !( hStoreHandle = CertOpenStore(
    CERT_STORE_PROV_SYSTEM,
    0,
    NULL,
    CERT_SYSTEM_STORE_CURRENT_USER,
    CERT_STORE_NAME)))
{
       HandleError("The MY store could not be opened.");
}

//--------------------------------------------------------------------
//
//得到证书的上下文，此证书必须能访问签名者的私钥

if(pSignerCert = CertFindCertificateInStore(
    hStoreHandle,
    MY_TYPE,
    0,
    CERT_FIND_SUBJECT_STR,
    SIGNER_NAME,
    NULL))
{
    printf("The signer's certificate was found.\n");
}
else
{
     HandleError( "Signer certificate not found.");
}

//--------------------------------------------------------------------
//初始化签名结构

SigParams.cbSize = sizeof(CRYPT_SIGN_MESSAGE_PARA);
SigParams.dwMsgEncodingType = MY_TYPE;
SigParams.pSigningCert = pSignerCert;
SigParams.HashAlgorithm.pszObjId = szOID_RSA_MD5;
SigParams.HashAlgorithm.Parameters.cbData = NULL;
SigParams.cMsgCert = 1;
SigParams.rgpMsgCert = &pSignerCert;
SigParams.cAuthAttr = 0;
SigParams.dwInnerContentType = 0;
SigParams.cMsgCrl = 0;
SigParams.cUnauthAttr = 0;
```

```
SigParams.dwFlags = 0;
SigParams.pvHashAuxInfo = NULL;
SigParams.rgAuthAttr = NULL;

//--------------------------------------------------------------------
//
// 首先得到 BLOB 的大小

if(CryptSignMessage(
      &SigParams,                  // Signature parameters
      FALSE,                       // Not detached
      1,                           // Number of messages
      MessageArray,                // Messages to be signed
      MessageSizeArray,            // Size of messages
      NULL,                        // Buffer for signed message
      &cbSignedMessageBlob)) // Size of buffer
{
      printf("The size of the BLOB is %d.\n",cbSignedMessageBlob);
}
else
{
      HandleError("Getting signed BLOB size failed");
}

//--------------------------------------------------------------------
// 分配 BLOB 的内存.

if(!(pbSignedMessageBlob =
     (BYTE*)malloc(cbSignedMessageBlob)))
{
      HandleError("Memory allocation error while signing.");
}

//--------------------------------------------------------------------
//

if(CryptSignMessage(
         &SigParams,                  //
         FALSE,                       //
         1,                           // 消息数量
         MessageArray,                // 待签名的消息
         MessageSizeArray,            // 消息大小
         pbSignedMessageBlob,     // 缓冲区
         &cbSignedMessageBlob)) // 缓冲区大小
{
```

```
        printf("The message was signed successfully. \n");
}
else
{
        HandleError("Error getting signed BLOB");
}

//--------------------------------------------------------------------
// 验证签名信息
//--------------------------------------------------------------------
// 初始化 VerifyParams 结构.

VerifyParams.cbSize = sizeof(CRYPT_VERIFY_MESSAGE_PARA);
VerifyParams.dwMsgAndCertEncodingType = MY_TYPE;
VerifyParams.hCryptProv = 0;
VerifyParams.pfnGetSignerCertificate = NULL;
VerifyParams.pvGetArg = NULL;

//--------------------------------------------------------------------
//

if(CryptVerifyMessageSignature(
        &VerifyParams,                  //.
        0,                               //
        pbSignedMessageBlob,            //.
        cbSignedMessageBlob,            //
        NULL,                            //
        &cbDecodedMessageBlob,                              //.
        NULL))                           // Pointer to signer certificate.
{
        printf("%d bytes need for the buffer.\n",cbDecodedMessageBlob);
}
else
{
        printf("Verification message failed. \n");
}

//--------------------------------------------------------------------
// 为缓冲区分配内存.

if(!(pbDecodedMessageBlob =
    (BYTE*)malloc(cbDecodedMessageBlob)))
{
        HandleError("Memory allocation error allocating decode BLOB.");
}
```

```
//--------------------------------------------------------------------
//
//   得到缓冲区的大小

if(CryptVerifyMessageSignature(
      &VerifyParams,                // Verify parameters.
      0,                             // Signer index.
      pbSignedMessageBlob,          // Pointer to signed BLOB.
      cbSignedMessageBlob,          // Size of signed BLOB.
      pbDecodedMessageBlob,         // Buffer for decoded message.
      &cbDecodedMessageBlob,        // Size of buffer.
      NULL))                         // Pointer to signer certificate.
{
      printf("The verified message is \n-> %s \n",pbDecodedMessageBlob);
}
else
{
      printf("Verification message failed. \n");
}

//--------------------------------------------------------------------
//

if(pbSignedMessageBlob)
        free(pbSignedMessageBlob);
if(pbDecodedMessageBlob)
        free(pbDecodedMessageBlob);
if(pSignerCert)
        CertFreeCertificateContext(pSignerCert);
if(CertCloseStore(
          hStoreHandle,
          CERT_CLOSE_STORE_CHECK_FLAG))
{
      printf("The store closed and all certificates are freed. \n");
}
else
{
      printf("Store closed after signing -- \n"
              "not all certificates, CRLs or CTLs were freed");
}
…
```

4.2 证书和证书库函数

这组函数管理、使用和取得证书、证书撤销列表和证书信任列表。这些函数可以分成一下几组：

4.2.1 证书库函数

一个用户站点可以收集许多证书。这些证书是为这个站点的用户所使用的，证书描述了这个用户的具体身份。对于每个人，可能有一个以上的证书。证书库和其相关的函数提供了对库获得、枚举、验证和使用证书库里的信息。

以下就是这些函数：

CertAddStoreToCollection	在证书库中增加一个证书
CertCloseStore	关闭一个证书库句柄
CertControlStore	如果证书缓冲区和证书本身内容不相符时，允许给应用程序发一个通知
CertDuplicateStore	通过增加引用计数来复制证书库句柄
CertEnumPhysicalStore	对于指定系统库枚举物理库
CertEnumSystemStore	枚举所有可用的系统库
CertEnumSystemStoreLocation	枚举可用系统库的所有位置
CertGetStoreProperty	得到一个库的属性
CertOpenStore	使用指定库类型来打开证书库
CertOpenSystemStore	打开一个系统证书库
CertRegisterPhysicalStore	在一个注册系统库里增加一个物理库
CertRegisterSystemStore	注册一个系统库
CertRemoveStoreFromCollection	从一个库集合里删除证书库
CertSaveStore	保存证书库
CertSetStoreProperty	设置证书属性
CertUnregisterPhysicalStore	从系统库中删除一个物理库
CertUnregisterSystemStore	反注册一个指定系统库

4.2.2 维护函数

CryptoAPI 提供了证书和证书库函数如下：

CertAddSerializeElementToStore	在库中增加一系列证书或 CRL
CertCreateContext	从编码字节中创建指定上下文
CertEnumSubjectInSortedCTL	在 CTL 库中枚举信任主题
CertFindSubjectInCTL	在 CTL 中寻找指定主题
CertFindSubjectInSortedCTL	在分类 CTL 中寻找指定主题

4.2.3 证书函数

下列函数是针对于证书的。大多数函数都是处理 CRL 和 CTL 的。

CertAddCertificateContextToStore	在证书库里增加一个证书上下文
CertAddCertificateLinkToStore	在证书库里增加一个对不同库里的证书上下文的链接
CertAddEncodedCertificateToStore	把编码证书转换成证书上下文并且把它加到证书库里
CertCreateCertificateContext	从编码证书中创建一个证书上下文。但这个上下文并不放到证书库里
CertCreateSelfSignCertificate	创建一个自签名证书
CertDeleteCertificateFromStore	从证书库里删除一个证书
CertDuplicateCertificate	通过增加引用计数来复制证书上下文
CertEnumCertificateInStore	在证书库里枚举证书上下文
CertFindCertificateInStore	在证书库里寻找证书上下文
CertFreeCertificateContext	释放一个证书上下文
CertGetIssuerCertificateFromStore	在证书库里得到指定主题证书的发行者
CertGetSubjectCertificateFromStore	获得主题证书的上下文
CertGetValidUsages	返回所有证书的用法
CertSerializeCertificateStoreElement	串行化编码证书的证书上下文
CertVerifySubjectCertificateContext	使用发行者来验证主题证书
CryptUIDlgViewContext	显示证书、CRL 或 CTL
CryptUIDlgSelectCertificateFromStore	从指定库中显示对话框，可以从中选择证书

4.2.4 证书撤销列表函数

CertAddCRLContextToStore	在证书库里增加一个 CRL 上下文
CertAddCRLLinkToStore	在不同的库里增加一个 CRL 上下文链接
CertAddEncodedCRLToStore	把编码 CRL 转化成 CRL 上下文然后把它加入到证书库中
CertCreateCRLContext	从编码 CRL 中创建 CRL 句柄，但不把它加到库中
CertDeleteCRLFromStore	从证书库里删除一个 CRL
CertDuplicateCRLContext	通过增加引用计数来复制 CRL 上下文
CertEnumCRLsInStore	枚举库里的 CRL 句柄
CertFindCertificateInCRL	从指定证书里寻找 CRL 列表
CertFindCRLInStore	在库里寻找 CRL 上下文
CertFreeCRLContext	释放 CRL 上下文
CertGetCRLFromStore	从库里得到 CRL 上下文句柄
CertSerializeCRLStoreElement	串行化 CRL 上下文的编码 CRL 和它的属性

4.2.5 证书信任列表函数

CertAddCTLContextToStore	把一个 CTL 上下文加入到证书库里
CertAddCTLLinkToStore	给不同库里的 CRL 上下文添加链接
CertAddEncodedCTLToStore	把编码 CTL 转化成 CTL 上下文并且把它加到证书库里
CertCreateCTLContext	从编码 CTL 中创建 CTL 上下文
CertDeleteCTLFromStore	从证书库里删除 CTL
CertDuplicateCTLContext	通过增加引用计数来复制 CTL 上下文
CertEnumCTLsInStore	在证书库里枚举 CTL 上下文
CertFindCTLInStore	在证书库里查找 CTL 上下文
CertFreeCTLContext	释放 CTL 上下文
CertSerializeCTLStoreElement	串行化 CTL 上下文的编码 CTL 和属性

4.2.6 扩展属性函数

CertEnumCertificateContextProperties	枚举指定证书上下文的属性
CertEnumCRLContextProperties	枚举指定 CRL 上下文的属性
CertEnumCTLContextProperties	枚举指定 CTL 上下文的属性
CertGetCertificateContextProperty	得到证书属性
CertGetCRLContextProperty	得到 CRL 属性
CertGetCTLContextProperty	得到 CTL 属性
CertSetCertificateContextProperty	设置证书属性
CertSetCRLContextProperty	设置 CRL 属性
CertSetCTLContextProperty	设置 CTL 属性

4.2.7 函数详解

4.2.7.1 打开/关闭系统证书库

4.2.7.1.1 CertOpenSystemStore

```
HCERTSTORE WINAPI CertOpenSystemStore(
  HCRYPTPROV hProv,
  LPCTSTR szSubsystemProtocol ，
);
```

参数：

hProv

[in] CSP 句柄。如果为 NULL，就为却省 CSP。如果不为 NULL，它必须是由 CryptAcquireContext 得到的 CSP 句柄。

szSubsystemProtocol

[in]系统证书库的名称。可以为”CA”、”MY”、”ROOT”、”SPC”。

说明：

此函数用来打开通用的系统证书库。

4.2.7.1.2 CertCloseStore

```
BOOL WINAPI CertCloseStore(
  HCERTSTORE hCertStore,
  DWORD dwFlags
);
```

参数：

hCertStore

[in]证书库句柄。

dwFlags

[in]典型地，此参数为 0。却省就是关闭证书库，对于为上下文分配的内存并不释放。如果想要检查并且释放所有为证书、CRL 和 CTL 上下文的分配的内存，就要置下列标志。

CERT_CLOSE_STORE_CHECK_FLAG	检查没有释放的证书、CRL 和 CTL 上下文。
CERT_CLOSE_STORE_FORCE_FLAG	强制释放所有和证书库相关的上下文。

说明：

此函数释放证书库句柄。

```
//--------------------------------------------------------------------
…
HCERTSTORE    hSystemStore;
if(hSystemStore = CertOpenSystemStore(0,"MY"))
{
  printf("The MY system store is open. Continue.\n");
  CertCloseStore(hSystemStore, CERT_CLOSE_STORE_CHECK_FLAG);
}
else
{
  printf("The MY system store did not open.\n");
  exit(1);
}
…
```

4.3 证书验证函数

证书验证是通过 CTL 和证书列表进行的。

4.3.1 使用 CTL 的函数

CertVerifyCTLUsage	验证 CTL 用法
CryptMsgEncodeAndSignCTL	编码和验证 CTL
CryptMsgGetAndVerifySigner	从一个消息中获得和验证 CTL
CryptMsgSignCTL	对包含 CTL 的消息进行签名

4.3.2 证书链验证函数

CertCreateCertificateChainEngine	为应用程序创建一个新的非却省的链引擎
CertCreateCTLEntryFromCertificateContextProperties	创建一个 CTL 入口
CertDuplicateCertificateChain	通过增加引用计数来复制证书链
CertFindChainInStore	在证书库里查找证书链
CertFreeCertificateChain	释放证书链
CertFreeCertificateChainEngine	释放证书链引擎
CertGetCertificateChain	从最后一个证书建立一个上下文链表
CertSetCertificateContextPropertiesFromCTLEntry	通过 CTL 入口属性来设置证书上下文的属性
CertIsValidCRLForCertificate	通过检查 CRL 来确定 CRL 是否包括指定被撤销的证书
CertVerifyCertificateChainPolicy	通过检查证书链来确定它的完整性

4.4 消息函数

CryptoAPI 消息函数包括两组：低级消息函数和简化消息函数。

低级消息函数直接和 PKCS#7 消息工作。这些函数对传输的 PKCS#7 数据进行编码，对接收到的 PKCS#7 数据进行解码，并且对接收到的消息进行解密和验证。

简化消息函数是比较高级的函数，是对几个低级消息函数和证书函数的封装，用来执行指定任务。这些函数在完成一个任务时，减少了函数调用的数量，因此简化了 CryptoAPI 的使用。

4.4.1 低级消息函数

CryptMsgCalculateEncodedLength	计算加密消息的长度
CryptMsgClose	关闭加密消息的句柄
CryptMsgControl	执行指定的控制函数
CryptMsgCountersign	标记消息中已存在的签名
CryptMsgCountersignEncoded	标记已存在的签名
CryptMsgDuplicate	通过增加引用计数来复制加密消息句柄

CryptMsgGetParam	对加密消息进行编码或者解码后得到的参数
CryptMsgOpenToDecode	打开加密消息进行解码
CryptMsgOpenToEncode	打开加密消息进行编码
CryptMsgUpdate	更新加密消息的内容
CryptMsgVerifyCountersignatureEncoded	验证 SignerInfo 结构中标记时间
CryptMsgVerifyCountersignatureEncodedEx	验证 SignerInfo 结构中标记时间，签名者可以是 CERT_PUBLIC_KEY_INFO 结构

4.4.2 简化消息函数

CryptDecodeMessage	对加密消息进行解码
CryptDecryptAndVerifyMessageSignature	对指定消息进行解密并且验证签名者
CryptDecryptMessage	解密指定消息
CryptEncryptMessage	加密指定消息
CryptGetMessageCertificates	返回包含消息的证书和 CRL 的证书库
CryptGetMessageSignatureCount	返回签名消息的签名者数量
CryptHashMessage	创建消息的哈希
CryptSignAndEncryptMessage	对消息进行签名并且加密
CryptSignMessage	对消息进行签名
CryptVerifyDetachedMessageHash	验证包含已解邦定哈希的哈希消息
CryptVerifyDetachedMessageSignature	验证包含已解邦定签名的签名消息
CryptVerifyMessageHash	验证一个哈希消息
CryptVerifyMessageSignature	验证一个签名消息

4.5 辅助函数

4.5.1 数据管理函数

CertCompareCertificate	比较两个证书是否相同
CertCompareCertificateName	通过比较两个证书名称来决定他们是否相同
CertCompareIntegerBlob	比较两个整数 BLOB
CertComparePublicKeyInfo	通过比较两个证书公钥来决定他们是否相同
CertFindAttribute	通过 OID 来查找属性
CertFindExtension	通过 OID 来查找扩展
CertFindRDNAttr	通过 OID 来查找 RDN 属性
CertGetIntendedKeyUsage	从证书中取得相关密钥用法
CertGetPublicKeyLength	从公钥 BLOB 中取得公钥/私钥长度
CertIsRDNAttrsInCertificateName	通过指定 RDN 数组属性比较证书名称属性来决定证书是否已包含了所有属性
CertVerifyCRLRevocation	验证主题证书是否在 CRL 中

CertVerifyCRLTimeValidity	验证 CRL 的有效时间
CertVerifyRevocation	验证主题证书是否在 CRL 中
CertVerifyTimeValidity	验证 CRL 的有效时间
CertVerifyValidityNesting	验证主题时间的有效性是否在发行者有效时间内
CryptExportPublicKeyInfo	导出公钥信息
CryptExportPublicKeyInfoEx	导出公钥信息（用户可以指定算法）
CryptFindCertificateKeyProvInfo	枚举 CSP 和它的密钥容器来查找对应于公钥的相应私钥
CryptFindLocalizedName	查找指定名字的局部化名称
CryptHashCertificate	哈希证书内容
CryptHashPublicKeyInfo	计算公钥信息的哈希
CryptHashToBeSigned	计算签名内容的信息哈希值
CryptImportPublicKeyInfo	把公钥信息导入 CSP 并且返回它的句柄
CryptImportPublicKeyInfoEx	把公钥信息导入 CSP 并且返回它的句柄
CryptMemAlloc	分配内存
CryptMemFree	释放内存
CryptMemRealloc	重新分配内存
CryptQueryObject	得到 BLOB 或文件的内容信息
CryptSignAndEncodeCertificate	对信息进行签名并且编码
CryptSignCertificate	对证书进行签名
CryptVerifyCertificateSignature	使用公钥信息对主题证书或 CRL 的签名进行验证
CryptVerifyCertificateSignatureEx	使用公钥信息对主题证书或 CRL 的签名进行验证

4.5.2 数据转换函数

CertAlgIdToOID	把 CSP 算法标示符转换成 OID
CertGetNameString	得到证书的主题或颁发者名称并且把它转换成字符串
CertNameToStr	把证书名称 BLOB 转换成字符串
CertOIDToAlgId	把 OID 转换成 CSP 算法表示符
CertRDNValueToStr	把名称值转换成字符串
CertStrToName	把字符串转换成编码证书名称
CryptBinaryToString	把二进制序列转换成字符串
CryptFormatObject	格式化编码数据，返回 Unicode 字符串
CryptStringToBinary	把格式化的字符串转换成二进制序列

4.5.3 增强密钥用法函数

CertAddEnhancedKeyUsageIdentifier	在证书 EKU 属性中增加一个用法标示符
CertGetEnhancedKeyUsage	获得证书的 EKU 扩展或属性信息
CertRemoveEnhancedKeyUsageIdentifier	从证书 EKU 扩展属性中删除用法标示符 OID

CertSetEnhancedKeyUsage	设置证书的 EKU 属性

4.5.4 密钥标示函数

CryptCreateKeyIdentifierFromCSP	创建 CSP 公钥的密钥标示符
CryptEnumKeyIdentifierProperties	枚举标示符和其属性
CryptGetKeyIdentifierProperty	从指定密钥标示符中获得指定属性
CryptSetKeyIdentifierProperty	设置指定密钥标示符的属性

4.5.5 证书库回调函数

CertDllOpenStoreProv	定义库提供者打开函数
CertStoreProvCloseCallback	决定当证书库引用计数为 0 时将发生的动作
CertStoreProvDeleteCertCallback	决定当从证书库中删除一个证书之前的动作
CertStoreProvDeleteCRLCallback	决定当从证书库中删除一个 CRL 之前的动作
CertStoreProvReadCertCallback	保留
CertStoreProvReadCRLCallback	保留
CertStoreProvSetCertPropertyCallback	决定在 CertSetCertificateContextProperty 和 CertGetCertificateContext 调用之前的动作
CertStoreProvSetCRLPropertyCallback	决定在 CertSetCRLContextProperty 和 CertGetCRLContextProperty 调用之前的动作
CertStoreProvWriteCertCallback	决定在证书库中加入一个证书前的动作
CertStoreProvWriteCRLCallback	决定在证书库中加入一个 CRL 前的动作
CertStoreProvReadCTL	读 CSP 的 CTL 上下文
CertStoreProvWriteCTL	决定 CTL 是否可被加入到证书库中
CertStoreProvDeleteCTL	决定 CTL 是否可被删除
CertStoreProvSetCTLProperty	决定是否可以设置 CTL 的属性
CertStoreProvControl	当缓冲库和存储库不同时，通知应用程序
CertStoreProvFindCert	在证书库中查找下一个证书
CertStoreProvFreeFindCert	释放前一个找到的证书上下文
CertStoreProvGetCertProperty	得到指定的证书属性
CertStoreProvFindCRL	查找第一个或下一个匹配的 CRL
CertStoreProvFreeFindCRL	释放前一个找到的 CRL 上下文
CertStoreProvGetCRLProperty	得到指定 CRL 属性
CertStoreProvFindCTL	查找第一个或下一个匹配的 CTL
CertStoreProvFreeFindCTL	释放前一个找到的 CTL 上下文
CertStoreProvGetCTLProperty	得到指定 CTL 属性

4.5.6 OID 支持函数

CryptEnumOIDFuction	枚举由编码类型、函数名和 OID 指定注册的 OID 函数
CryptEnumOIDInfo	枚举注册的 OID 信息

CryptEnumOIDInfo	使用指定的密钥和组查找 OID 信息
CryptFreeOIDFuctionAddress	释放 OID 函数地址句柄
CryptGetDefaultOIDDllList	对于指定的函数结合和类型获得却省注册的 DLL 入口
CryptGetDefaultOIDFuctionAddress	获得已安装的第一次或下一个却省函数或者加载包含却省函数的 DLL
CryptGetOIDFuctionAddress	搜索匹配指定编码类型和 OID 函数列表，如果没有找到，就查找注册表。
CryptGetOIDFuctionValue	获得指定编码类型、函数名称和 OID 的值
CryptInitOIDFuctionSet	初始化 OID 函数集合的句柄
CryptInstallOIDFuctionAddress	安装可调用的 OID 函数地址集合
CryptRegisterDefaultOIDFuction	注册包含却省函数的 DLL
CryptRegisterOIDFuction	注册包含指定函数的 DLL
CryptRegisterOIDInfo	注册由 CRYPT_OID_INFO 指定的 OID 信息
CryptSetOIDFuctionValue	设置编码类型、函数名称等的值
CryptUnregisterDefaultOIDFunction	卸载包含却省函数的 DLL
CryptUnregisterOIDFuction	卸载包含函数的 DLL
CryptUnregisterOIDInfo	卸载指定 OID 的信息

4.5.7 远程对象恢复函数

CryptGetObjectUrl	从证书、CTL 或 CRL 中取得远程对象的 URL
CryptRetrieveObjectByUrl	由 URL 指定位置恢复 PKI 对象

4.5.8 PFX 函数

PFXExportCertStore	从证书库中导出证书或证书和私钥
PFXExportCertStoreEx	从证书库中导出证书或证书和私钥
PFXImportCertStore	从 PFX BLOB 导入到指定证书库
PFXIsPFXBlob	把外层 BLOB 像 pfx 包那样解码
PFXVerifyPassword	把外层 BLOB 像 pfx 包那样解码,并且用指定口令解密

5 待续

由于当时写此文档的时候 CAPI 的版本为 1.0，现在已经发展到 2.0，CAPI 增加了不少函数。所以我会再以后工作之余抽空把 2.0 新加的函数在文档中补充进去，也可能添加 CAPICOM 的内容。

谢谢！