```
BCRYPT_HASH_HANDLE          hHash          = NULL;
NTSTATUS                    Status         = STATUS_UNSUCCESSFUL;
SECURITY_STATUS             secStatus      = ERROR_SUCCESS;
DWORD                       cbHash         = 0,
                            cbData         = 0,
                            cbHashObject   = 0;
PBYTE                       pbHashObject   = NULL;
PBYTE                       pbHash         = NULL,
                            pbBlob         = NULL;
PBYTE                       pbSignature    = NULL;
DWORD                       cbSignature    = 0,
                            cbBlob         = 0;
BCRYPT_PKCS1_PADDING_INFO   PKCS1PaddingInfo= {0};
// 打开算法提供者句柄
if(!NT_SUCCESS(Status = BCryptOpenAlgorithmProvider(
    &hAlgorithm,    NCRYPT_SHA1_ALGORITHM,
    NULL, 0)))
{
    wprintf(L"Error 0x%x returned by BCryptOpenAlgorithmProvider\n", Status);
    goto Cleanup;
}
if(!NT_SUCCESS(Status = BCryptGetProperty(
    hAlgorithm,   BCRYPT_OBJECT_LENGTH,
    (PBYTE)&cbHashObject,   sizeof(DWORD),   &cbData,    0)))
{
    wprintf(L"**** Error 0x%x returned by BCryptGetProperty\n", Status);
    goto Cleanup;
}
pbHashObject = (PBYTE)HeapAlloc (GetProcessHeap (), 0,cbHashObject);
if(NULL == pbHashObject)
{
    wprintf(L"**** memory allocation failed\n");
    goto Cleanup;
}
//获取 HASH 缓冲区空间大小
if(!NT_SUCCESS(Status = BCryptGetProperty(
    hAlgorithm,   BCRYPT_HASH_LENGTH,
    (PBYTE)&cbHash,   sizeof(DWORD),   &cbData,      0)))
{
    wprintf(L"**** Error 0x%x returned by BCryptGetProperty\n", Status);
    goto Cleanup;
}
pbHash = (PBYTE)HeapAlloc (GetProcessHeap (), 0, cbHash);
```

```
if(NULL == pbHash)
{
    wprintf(L"**** memory allocation failed\n");
    goto Cleanup;
}
if(!NT_SUCCESS(Status = BCryptCreateHash(
    hAlgorithm,    &hHash,    pbHashObject,
    cbHashObject,    NULL,    0,    0)))
{
    wprintf(L"**** Error 0x%x returned by BCryptCreateHash\n", Status);
    goto Cleanup;
}
if(!NT_SUCCESS(Status = BCryptHashData(
    hHash,    (PBYTE)KNOWN_DATA,
    KNOWN_DATALEN,    0)))
{
    wprintf(L"**** Error 0x%x returned by BCryptHashData\n", Status);
    goto Cleanup;
}
if(!NT_SUCCESS(Status = BCryptFinishHash(
    hHash,    pbHash,    cbHash,    0)))
{
    wprintf(L"**** Error 0x%x returned by BCryptFinishHash\n", Status);
    goto Cleanup;
}
PKCS1PaddingInfo.pszAlgId = NCRYPT_SHA1_ALGORITHM;
if(FAILED(secStatus = NCryptSignHash(
    hKey,    &PKCS1PaddingInfo,    pbHash,    cbHash,
    NULL,    0,    &cbSignature,    NCRYPT_PAD_PKCS1_FLAG)))
{
    wprintf(L"**** Error 0x%x returned by NCryptSignHash\n", secStatus);
    goto Cleanup;
}
pbSignature = (PBYTE)HeapAlloc (GetProcessHeap (), 0, cbSignature);
if(NULL == pbSignature)
{
    wprintf(L"**** memory allocation failed\n");
    goto Cleanup;
}
if(FAILED(secStatus = NCryptSignHash(
    hKey,    &PKCS1PaddingInfo,    pbHash,    cbHash,
    pbSignature,    cbSignature,    &cbSignature,
    NCRYPT_PAD_PKCS1_FLAG)))
```

```
        {
            wprintf(L"**** Error 0x%x returned by NCryptSignHash\n", secStatus);
            goto Cleanup;
        }
    Cleanup:
        if(hHash) {     BCryptDestroyHash(hHash);       }
        if(hAlgorithm) {    BCryptCloseAlgorithmProvider(hAlgorithm,0);     }
        if(pbHashObject) {    HeapFree(GetProcessHeap(), 0, pbHashObject);     }
        if(pbHash) {      HeapFree(GetProcessHeap(), 0, pbHash);     }
        if(pbSignature) {     HeapFree(GetProcessHeap(), 0, pbSignature);     }
        if(pbBlob) {      HeapFree(GetProcessHeap(), 0, pbBlob);     }
        return;
    }
```

12.5 PC/SC

12.5.1 PC/SC 简介

PC/SC 即个人计算机（Personal Computer）/智能卡（Smart Card），由微软公司与世界其他著名的智能卡厂商组成的 PC/SC 工作组提出，它是为智能卡访问 Windows 平台定义的一种标准结构。

PC/SC 标准是一个基于 Windows 平台的标准用户接口（API），提供了一个从个人计算机（Personal Computer）到智能卡（Smart Cart）的整合环境，为集成电路卡（ICC）与个人计算机系统设计的交互规范，让智能卡进入 PC 世界变得容易。PC/SC 的主要优点就是让应用程序不必为了与智能卡通信而去了解智能卡读卡器的细节，而且应用程序还能适用于任何遵从 PC/SC 标准的读卡器。

Windows 系统中，PC/SC 功能通过 Windows 智能卡（WinSCard）客户端 API 提供给应用程序，该 API 在 winscard.dll 和 scarddlg.dll 中实现。

由于智能卡是一种共享资源，在任何给定时间都可能有多个应用程序试图与该卡进行通信，与该卡的通信必须是串行的，因此要使用资源管理器模型。资源管理器强制采用简化的、类似于数据库的锁定模型。在 PC/SC 中，持有给定智能卡读卡器（也称为接口设备或 IFD）的锁，就等于掌握了智能卡资源。

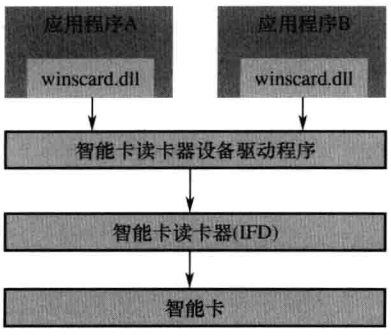

图 12-9　智能卡调用分层结构

图 12-9 说明了 PC/SC 组件堆栈。PC/SC 客户端代码是在应用程序进程中加载的。资源管理器是一种系统服务，是独立于客户端的。读卡器设备驱动程序是唯一在内核模式下运行的 PC/SC 堆栈组件。读卡器驱动程序被资源管理器独占加载，以防止应用程序忽略事务模型。

智能卡在 Windows 中的典型用途通常与身份验证相关。例如，它们可以代替密码进行登录。这些智能卡身份

验证方案是基于加密技术的，这就导致一直以来，在 Windows 中添加对新型智能卡的支持始终要求供应商部署一个 CSP（加密服务提供商）或 CNG（下一代加密接口）的插件，该插件实现了 CryptoAPI 1.0 或 CAPI 2.0 的接口。

为了简化 CSP/CNG 插件的编写，微软提供了智能卡的微型驱动（mini-driver）接口，通过编写 mini-driver，应用程序可以调用基于微软标准的 Microsoft Smart Card Base CSP 和 KSP 服务使用智能卡。图 12-10 说明了智能卡的 mini-driver 与基于 CSP 的应用程序的组成结构。应用程序通过 CAPI 接口，调用 Smart Card Base Cryptographic 服务提供者，智能卡 CSP 提供者使用智能卡 mini-driver 与智能卡设备交互，完成密码运算，而 mini-driver 使用 WinScard 接口与智能卡交互。从图中可以看出，通过智能卡 mini-driver 驱动，应用程序可以完全利用 CSP/CNG 提供的接口完成与智能卡的交互。

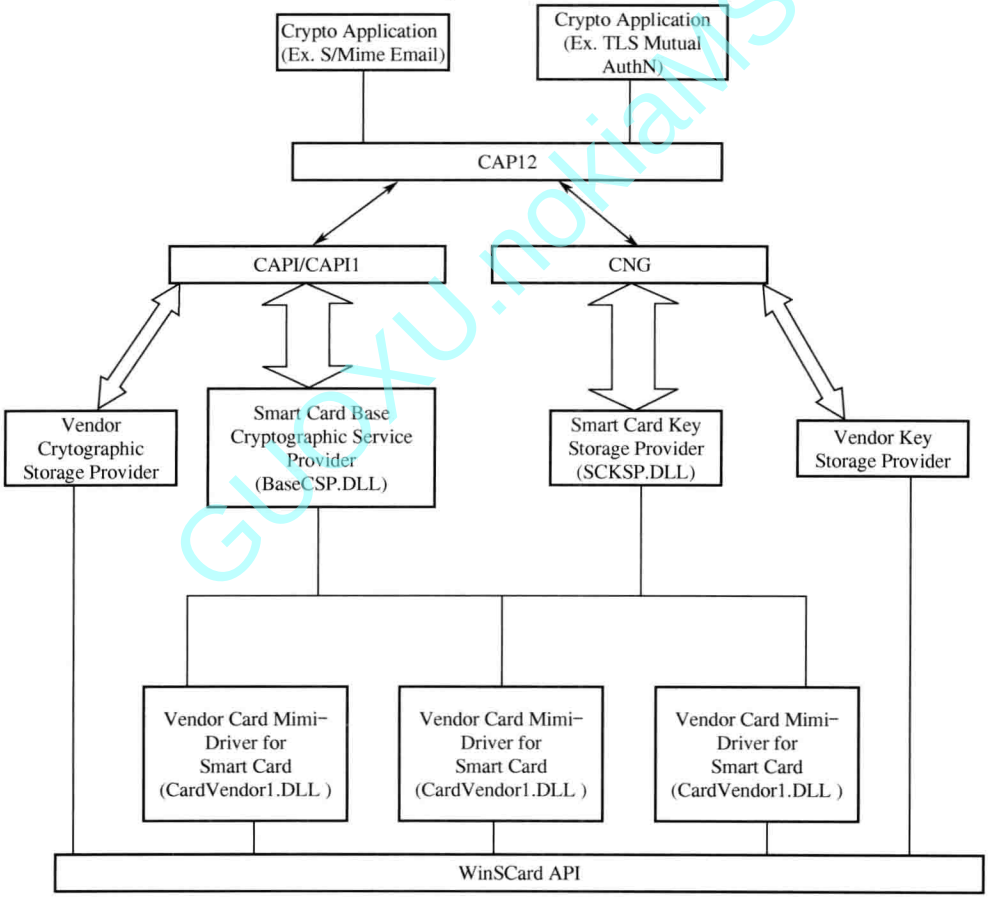

图 12-10　智能卡应用与 mini-driver 的交互结构

如果不使用 CSP/CNG 提供的接口，应用程序调用智能卡功能必须通过发送指令形式，使用 SCardTransmit 函数发送指令，具体示例代码如下：

```
LONG rv;
DWORD dwRecvLength;
```

```
BYTE pbRecvBuffer[258];
BYTE cmd2[] = { 0x00, 0x00, 0x00, 0x00 };
dwRecvLength = sizeof(pbRecvBuffer);
rv = SCardTransmit(hCard, &pioSendPci, cmd2, sizeof(cmd2), NULL,
                   pbRecvBuffer, &dwRecvLength);
```

12.5.2 使用证书

为了便于说明 winscard 接口和 CryptoAPI 接口的联合调用，Windows Platform SDK 中自带一个代码示例，把智能卡证书导入到证书库。

在该示例中，首先通过 SCardEstablishContext 获得资源管理控制器上下文，接着调用 SCardListReaders 列出所有的已安装的读卡器，然后调用自定义函数 SCardPropCert 完成证书导入。SCardPropCert 返回后，调用 SCardFreeMemory 释放申请的内存，调用 SCardReleaseContext 释放资源管理控制器上下文。

在 SCardPropCert 函数中，执行如下操作：

① 获取读卡器的数量。

② 分配存储读卡器状态数据数组，用于获取读卡器信息。

③ 调用函数 SCardGetStatusChange 获取读卡器状态。

④ 通过 SCARD_READERSTATE 的 dwEventState 字段，判断 SCARD_STATE_PRESENT 位是否被置位，若置位表示存在智能卡读卡器。

⑤ 对存在的读卡器，调用 SCardListCards 函数获取智能卡名称。

⑥ 调用函数 SCardGetCardTypeProviderName 获取智能卡 CSP 提供者名称。

⑦ 分配容器名称存储空间内存，然后构建容器名称。

⑧ 调用函数 CryptAcquireContext 打开 CSP 提供者上下文句柄。

⑨ 调用自定义函数 CryptPropCert 完成证书导出。

⑩ 调用 CryptReleaseContext 释放 CSP 提供者上下文句柄。

⑪ 释放申请的内存空间。

在 CryptPropCert 函数中，执行如下操作：

① 调用函数 CryptGetProvParam 获取密钥容器名称。

② 调用自定义函数 GetCert 分别获得签名私钥和加密私钥对应证书。

③ 调用自定义函数 AddCert 把证书加入个人证书库中。

④ 释放申请的内存空间。

在 GetCert 函数中，执行如下操作：

① 调用函数 CryptGetUserKey 获得容器中的密钥句柄。

② 调用函数 CryptGetKeyParam 获取证书数据长度。

③ 为证书数据分配内存空间。

④ 调用函数 CryptGetKeyParam 获取证书数据。

⑤ 释放申请内存空间。

⑥ 调用函数 CryptDestroyKey 释放获得证书句柄。

在 AddCert 函数中，执行如下操作：