① 调用函数 CertOpenStore 打开证书库句柄。

② 以获取的证书数据为参数，调用函数 CertCreateCertificateContext 生成证书对象。

③ 调用函数 CertSetCertificateContextProperty 设置证书的易用名、提供者名称、密钥类型、容器名。

④ 调用函数 CertAddCertificateContextToStore 把证书加入到证书库中。

⑤ 释放证书对象。

⑥ 关闭证书库句柄。

该示例代码如下：

```
#define _WIN32_WINNT 0x0400
#include <tchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winscard.h>
#include <wincrypt.h>
// Macros
#define MALLOC(size)    ((LPBYTE) LocalAlloc(LPTR, size))
#define FREE(buffer)    (LocalFree((LPBYTE) buffer))
// Prototypes
LONG SCardPropCert (IN SCARDCONTEXT hContext,
      IN LPCTSTR mszReaderNames,    IN LPCTSTR szStoreName);
LONG CryptPropCert (IN HCRYPTPROV hCryptProv,
      IN LPCTSTR szCSPName,    IN LPCTSTR szStoreName);
LONG GetCert (IN HCRYPTPROV hCryptProv,
      IN DWORD dwKeySpec,      OUT LPBYTE * lplpbCert,
      OUT DWORD * lpdwCertLength);
LONG AddCert (IN HCRYPTPROV hCryptProv,
      IN LPBYTE lpbCert,    IN DWORD dwCertLength,
      IN DWORD dwKeySpec,    IN LPCWSTR szCertFriendlyName,
      IN LPCWSTR zContainerName, IN LPCWSTR szCSPName,
      IN LPCWSTR szStoreName);
int __cdecl _tmain (int argc, _TCHAR    * argv[])
{
    LONG lResult;
    DWORD dwNumReaders = 0;
    SCARDCONTEXT hContext = NULL;
    LPTSTR mszReaderNames = NULL;
    __try {
        // Establish context with the resource manager.
        lResult = SCardEstablishContext(SCARD_SCOPE_USER,
                NULL,     NULL,     &hContext);
        if (lResult != SCARD_S_SUCCESS) {     __leave;      }
```

```
        // Get the list of reader(s) associated with the specified group(s).
        // Note: The buffer is automatically allocated and must be freed
        //          by SCardFreeMemory().
        DWORD dwAutoAllocate = SCARD_AUTOALLOCATE;
        lResult = SCardListReaders(hContext,   SCARD_DEFAULT_READERS,
                (LPTSTR) &mszReaderNames,   &dwAutoAllocate);
        if (lResult != SCARD_S_SUCCESS) {     __leave;      }
        // Propagate all digital certificate(s) found in all reader(s) to the
        // local "My" store.
        lResult = SCardPropCert(hContext, mszReaderNames, _T("My"));
    }
    __finally {
        LONG lReturn;
        // Don't forget to free resources, if allocated.
        if (mszReaderNames != NULL) {
            lReturn = SCardFreeMemory(hContext, (LPVOID) mszReaderNames);
            // If successful so far, then capture the return code
            // from SCFree(); otherwise, don't bother.
            if (lResult == SCARD_S_SUCCESS) {     lResult = lReturn;    }
        }
        if (hContext != NULL) {
            lReturn = SCardReleaseContext(hContext);
            // If successful so far, then capture the SCardReleaseContext()
            // return code; otherwise, don't bother.
            if (lResult == SCARD_S_SUCCESS) {    lResult = lReturn;      }
        }
    }
    // Inform user if an error had occurred.
    if (lResult != SCARD_S_SUCCESS) {
        _tprintf(_T("\nError [0x%x]: Program terminated abnormally.\n"), lResult);
    }
    return lResult;
}
LONG SCardPropCert (IN SCARDCONTEXT hContext,
     IN LPCTSTR mszReaderNames,      IN LPCTSTR szStoreName)
{
    LONG lResult;
    LPSCARD_READERSTATE lpReaderStates = NULL;
    // Make sure pointer parameters are not NULL.
    if (mszReaderNames == NULL || szStoreName == NULL) {
        return SCARD_E_INVALID_PARAMETER;
    }
    __try {
```

```
DWORD dwNumReaders;
LPCTSTR szReaderName;
// Count number of readers.
for (dwNumReaders = 0, szReaderName = mszReaderNames;
        *szReaderName != _T('\0');     dwNumReaders++) {
    szReaderName += lstrlen(szReaderName) + 1;
}
// Allocate memory for SCARD_READERSTATE array.
lpReaderStates = (LPSCARD_READERSTATE)
   MALLOC(dwNumReaders * sizeof(SCARD_READERSTATE));
if (lpReaderStates == NULL) {
    lResult = SCARD_E_NO_MEMORY;
    __leave;
}
// Prepare state array.
ZeroMemory((LPVOID) lpReaderStates,
            dwNumReaders * sizeof(SCARD_READERSTATE));
DWORD i;
for (i = 0, szReaderName = mszReaderNames;
       i < dwNumReaders;       i++) {
    lpReaderStates[i].szReader = (LPCTSTR) szReaderName;
    lpReaderStates[i].dwCurrentState = SCARD_STATE_UNAWARE;
    szReaderName += lstrlen(szReaderName) + 1;
}
// Initialize card status.
lResult = SCardGetStatusChange(hContext,    INFINITE,
        lpReaderStates,     dwNumReaders);
if (lResult != SCARD_S_SUCCESS) {       __leave;     }
// For each card found, find the proper CSP and propagate the
// certificate(s) to the specified local store.
for (i = 0; i < dwNumReaders && lResult == SCARD_S_SUCCESS; i++) {
    DWORD dwAutoAllocate;
    LPTSTR szCardName = NULL;
    LPTSTR szCSPName = NULL;
    LPTSTR szContainerName = NULL;
    HCRYPTPROV hCryptProv = NULL;
    __try {
        // Card in this reader?
        if (!(lpReaderStates[i].dwEventState & SCARD_STATE_PRESENT)){
            // No card in this reader.
            continue;
        }
        // Get card name.
```

```
        dwAutoAllocate = SCARD_AUTOALLOCATE;
        lResult = SCardListCards(hContext,  lpReaderStates[i].rgbAtr,
            NULL,  0,  (LPTSTR) &szCardName,  &dwAutoAllocate);
        if (lResult != SCARD_S_SUCCESS) {  __leave;  }
        // Get card's CSP name.
        dwAutoAllocate = SCARD_AUTOALLOCATE;
        lResult = SCardGetCardTypeProviderName(hContext,  szCardName,
            SCARD_PROVIDER_CSP,  (LPTSTR) &szCSPName,
            &dwAutoAllocate);
        if (lResult != SCARD_S_SUCCESS) {  __leave;  }
        // Prepare fully qualified container name.
        szContainerName = (LPTSTR) MALLOC((sizeof(_T("\\\\.\\")) +
            lstrlen(lpReaderStates[i].szReader) +
            sizeof(_T("\\0"))) * sizeof(TCHAR));
        if (szContainerName == NULL) {
            lResult = SCARD_E_NO_MEMORY;
            __leave;
        }
        wsprintf(szContainerName, _T("\\\\.\\%s\\"),  lpReaderStates[i].szReader);
        // Obtain the crypto context.
        // CRYPT_SILENT forces the CSP to raise no UI. The fully qualified
        // container name indicates which reader to connect to, so the
        // user should not be prompted to insert or select a card.
        if (!CryptAcquireContext(&hCryptProv,  szContainerName,
            szCSPName,  PROV_RSA_FULL,  CRYPT_SILENT)) {
            lResult = GetLastError();
            __leave;
        }
        // Propagate the cert.
        lResult = CryptPropCert(hCryptProv,  szCSPName,  szStoreName);
    }
    __finally {
        LONG lReturn;
        // Don't forget to free resources, if allocated.
        if (hCryptProv != NULL) {
            if (!CryptReleaseContext(hCryptProv, 0)) {
                if (lResult == SCARD_S_SUCCESS) {
                    lResult = GetLastError();
                }
            }
        }
        if (szContainerName != NULL) {
            FREE((LPVOID) szContainerName);
```

```
                }
                if (szCSPName != NULL) {
                    lReturn = SCardFreeMemory(hContext, (LPVOID) szCSPName);
                    if (lResult == SCARD_S_SUCCESS) {
                        lResult = lReturn;
                    }
                }
                if (szCardName != NULL)
                {
                    lReturn = SCardFreeMemory(hContext, (LPVOID) szCardName);
                    if (lResult == SCARD_S_SUCCESS) {
                        lResult = lReturn;
                    }
                }
            }
        }
    }
    __finally {
        // Don't forget to free resources, if allocated.
        if (lpReaderStates != NULL) {
            FREE((LPVOID) lpReaderStates);
        }
    }
    return lResult;
}
LONG CryptPropCert (IN HCRYPTPROV hCryptProv,
    IN LPCTSTR szCSPName,    IN LPCTSTR szStoreName)
{
    LONG lResult;
    LPCTSTR szContainerName = NULL;
    // Make sure pointer parameters are not NULL.
    if (szCSPName == NULL || szStoreName == NULL) {
        return SCARD_E_INVALID_PARAMETER;
    }
    __try {
        // Query length of key container name.
        DWORD cbContainerName = 0;
        if (!CryptGetProvParam(hCryptProv,    PP_CONTAINER,
                NULL,    // NULL to query container name length
                &cbContainerName,       0)) {
            lResult = GetLastError();
            __leave;
        }
```