```
// Allocate memory for key container name.
szContainerName = (LPCTSTR) MALLOC(cbContainerName);
// Now get the key container name.
if (!CryptGetProvParam(hCryptProv,    PP_CONTAINER,
        (PBYTE) szContainerName,    &cbContainerName, 0)) {
    lResult = GetLastError();
    __leave;
}
// For each key pair found in the smart card, store the corresponding
// digital certificate to the specified local store.
const DWORD rgdwKeys[] = {AT_KEYEXCHANGE, AT_SIGNATURE};
const DWORD cdwKeys = sizeof(rgdwKeys) / sizeof(rgdwKeys[0]);
for (DWORD i = 0; i < cdwKeys; i++) {
    DWORD dwCertLength = 0;
    LPBYTE lpbCert = NULL;
    LPWSTR wszCertFriendlyName = NULL;
    LPWSTR wszContainerName = NULL;
    LPWSTR wszCSPName = NULL;
    LPWSTR wszStoreName = NULL;
    __try {
        // Get the certificate data.
        lResult = GetCert(hCryptProv,    rgdwKeys[i],
                &lpbCert,    &dwCertLength);
        if (lResult != SCARD_S_SUCCESS) {
            if (lResult == NTE_NO_KEY) {
                // We are OK if there is no key of such type.
                // It means there is nothing to do.
                lResult = SCARD_S_SUCCESS;
            }
            __leave;
        }
        // Allocate memory for UNICODE strings.
        TCHAR szCertFriendlyName[] = "";
        DWORD cchCertFriendlyName = (lstrlen(szCertFriendlyName) + 1)
                                        * sizeof(WCHAR);
        DWORD cchContainerName = (lstrlen(szContainerName) + 1)
                                    * sizeof(WCHAR);
        DWORD cchCSPName = (lstrlen(szCSPName) + 1) * sizeof(WCHAR);
        DWORD cchStoreName = (lstrlen(szStoreName) + 1) * sizeof(WCHAR);
        wszCertFriendlyName = (LPWSTR) MALLOC(cchCertFriendlyName);
        wszContainerName = (LPWSTR) MALLOC(cchContainerName);
        wszCSPName = (LPWSTR) MALLOC(cchCSPName);
        wszStoreName = (LPWSTR) MALLOC(cchStoreName);
```

```
            if (wszCertFriendlyName == NULL || wszContainerName == NULL ||
                wszCSPName == NULL ||  wszStoreName == NULL) {
                lResult = SCARD_E_NO_MEMORY;
                __leave;
            }
            // Setup UNICODE strings.
#ifdef _UNICODE
            lstrcpy(wszCertFriendlyName, szCertFriendlyName);
            lstrcpy(wszContainerName, szContainerName);
            lstrcpy(wszCSPName, szCSPName);
            lstrcpy(wszStoreName, szStoreName);
#else
            mbstowcs(wszCertFriendlyName, szCertFriendlyName,
                    cchCertFriendlyName);
            mbstowcs(wszContainerName, szContainerName, cchContainerName);
            mbstowcs(wszCSPName, szCSPName, cchCSPName);
            mbstowcs(wszStoreName, szStoreName, cchStoreName);
#endif
            // Add the certificate to the specified local store.
            lResult = AddCert(hCryptProv,  lpbCert,
                dwCertLength,  rgdwKeys[i],  wszCertFriendlyName,
                wszContainerName,  wszCSPName,  wszStoreName);
            if (lResult != SCARD_S_SUCCESS) {
                __leave;
            } else {
                _tprintf(
                    _T("\nPropagated cert from %s to \"%s\" certificate store.\n"),
                    szCSPName,  szStoreName);
            }
        }
        __finally {
            // Don't forget to free resources, if allocated.
            if (lpbCert != NULL) {  FREE(lpbCert);  }
            if (wszCertFriendlyName != NULL) { FREE(wszCertFriendlyName); }
            if (wszContainerName != NULL) {  FREE(wszContainerName);  }
            if (wszCSPName != NULL) {  FREE(wszCSPName);  }
            if (wszStoreName != NULL) {  FREE(wszStoreName);  }
            if (lResult != SCARD_S_SUCCESS) {  __leave;  }
        }
    }
    __finally {
        // Don't forget to free resources, if allocated.
```

```
        if (szContainerName != NULL) { FREE(szContainerName);     }
    }
    return lResult;
}
LONG GetCert (IN HCRYPTPROV hCryptProv,    IN DWORD dwKeySpec,
     OUT LPBYTE * lplpbCert,     OUT DWORD * lpdwCertLength)
{
    LONG lResult = SCARD_S_SUCCESS;
    HCRYPTKEY hCryptKey = NULL;
    LPBYTE lpbCert = NULL;
    DWORD dwCertLength = 0;
    // Make sure pointer parameters are not NULL.
    if (lplpbCert == NULL || lpdwCertLength == NULL) {
        return SCARD_E_INVALID_PARAMETER;
    }
    __try {
        // Get key handle.
        if (!CryptGetUserKey(hCryptProv, dwKeySpec, &hCryptKey)) {
            lResult = GetLastError();
            __leave;
        }
        // Query certificate data length.
        if (!CryptGetKeyParam(hCryptKey,     KP_CERTIFICATE,
                NULL,    // NULL to query certificate data length
                &dwCertLength,     0)) {
            // We expect ERROR_MORE_DATA. If that's not the case, then
            // something is not right.
            lResult = GetLastError();
            if (lResult == ERROR_MORE_DATA) {
                lResult = SCARD_S_SUCCESS;
            } else {
                __leave;
            }
        }
        // Allocate memory for certificate data.
        lpbCert = (LPBYTE) MALLOC(dwCertLength);
        if (lpbCert == NULL) {
            lResult = SCARD_E_NO_MEMORY;
            __leave;
        }
        // Now read the certificate data.
        if (!CryptGetKeyParam(hCryptKey,    KP_CERTIFICATE,
           lpbCert,&dwCertLength, 0)) {
```

```
            lResult = GetLastError();
            __leave;
        }
    }
    __finally {
        // Don't forget to free resources, if allocated.
        if (lResult == SCARD_S_SUCCESS) {
            *lplpbCert = lpbCert;
            *lpdwCertLength = dwCertLength;
        } else if (lpbCert != NULL) {
            FREE(lpbCert);
        }
        if (hCryptKey != NULL) {
            if (!CryptDestroyKey(hCryptKey)) {
                if (lResult == SCARD_S_SUCCESS) {
                    lResult = GetLastError();
                }
            }
        }
    }
    return lResult;
}

LONG AddCert (IN HCRYPTPROV hCryptProv,
              IN LPBYTE lpbCert,
              IN DWORD dwCertLength,
              IN DWORD dwKeySpec,
              IN LPCWSTR wszCertFriendlyName,
              IN LPCWSTR wszContainerName,
              IN LPCWSTR wszCSPName,
              IN LPCWSTR wszStoreName)
{
    LONG lResult = SCARD_S_SUCCESS;
    HCERTSTORE hCertStore = NULL;
    PCCERT_CONTEXT pCertContext = NULL;
    // Make sure pointer parameters are not NULL.
    if (lpbCert == NULL ||    wszContainerName == NULL ||
        wszCSPName == NULL ||    wszStoreName == NULL) {
        return SCARD_E_INVALID_PARAMETER;
    }
    __try {
        // Open the user's specified store for writing.
        hCertStore = CertOpenStore(CERT_STORE_PROV_SYSTEM_W,
```

```
        0,     hCryptProv,
        CERT_STORE_NO_CRYPT_RELEASE_FLAG |
        CERT_SYSTEM_STORE_CURRENT_USER,
        wszStoreName);
if (NULL == hCertStore) {
    lResult = GetLastError();
    __leave;
}
// Build certificate context for this certificate.
pCertContext = CertCreateCertificateContext(X509_ASN_ENCODING,
        lpbCert,   dwCertLength);
if (pCertContext == NULL) {
    lResult = GetLastError();
    __leave;
}
// Add the friendly name, if provided.
if (wszCertFriendlyName != NULL) {
    CRYPT_DATA_BLOB DataBlob;
    ZeroMemory((PVOID)&DataBlob, sizeof(CRYPT_DATA_BLOB));
    DataBlob.cbData = (wcslen(wszCertFriendlyName) + 1) * sizeof(WCHAR);
    DataBlob.pbData = (LPBYTE) wszCertFriendlyName;
    if (!CertSetCertificateContextProperty(pCertContext,
            CERT_FRIENDLY_NAME_PROP_ID,      0,
            (const void *) &DataBlob)) {
        lResult = GetLastError();
        __leave;
    }
}
// Add the CSP & key container info. This is used by CAPI to load the
// CSP and find the keyset when the user indicates this certificate.
CRYPT_KEY_PROV_INFO KeyProvInfo;
ZeroMemory((PVOID)&KeyProvInfo, sizeof(CRYPT_KEY_PROV_INFO));
KeyProvInfo.pwszContainerName = (LPWSTR) wszContainerName;
KeyProvInfo.pwszProvName          = (LPWSTR) wszCSPName;
KeyProvInfo.dwProvType            = PROV_RSA_FULL;
KeyProvInfo.dwFlags               = 0;
KeyProvInfo.dwKeySpec             = dwKeySpec;
if (!CertSetCertificateContextProperty(pCertContext,
        CERT_KEY_PROV_INFO_PROP_ID,
        0,   (const void *) &KeyProvInfo)) {
    lResult = GetLastError();
    __leave;
}
```