Smartcard in 2008 and Vista..National ID card? No UPN? No EKU? No problem!

Samstag, 16. Juli 2016 08:10

I recently worked on an issue which combined a number of the things I enjoy about my job. New technology, interesting problem and satisfied customers.

Here was the problem.

A country in the EU is rolling out national identity cards. They want to use these cards for all sorts of transactions, including logging into private corp domains.

Previously this would have been impossible due to somewhat restrictive requirements for domain logon.

First, lets summarize the problems.

1. Since it is a government issued card, the private companies obviously have no control over the issuing CA's or the Root.

2. The cards have no UPN or subject alternate name.

3. The cards have no AT_KEYEXCHANGE key type.

4. The CRLs , may or may not be available.

5. The card does not specify the EKU \ App OID – 1.3.6.1.4.1.311.20.2.2 – which is Smart Card logon.

There is no way XP or 2003 would ever work for what they wanted.

However, put 2008 and Vista together and you have a great solution.

Here is an outline for the solution.

1. Create 2008 domain controller.

2. Join Vista SP1

3. Install Card Module

4. Configure a group policy as follows:

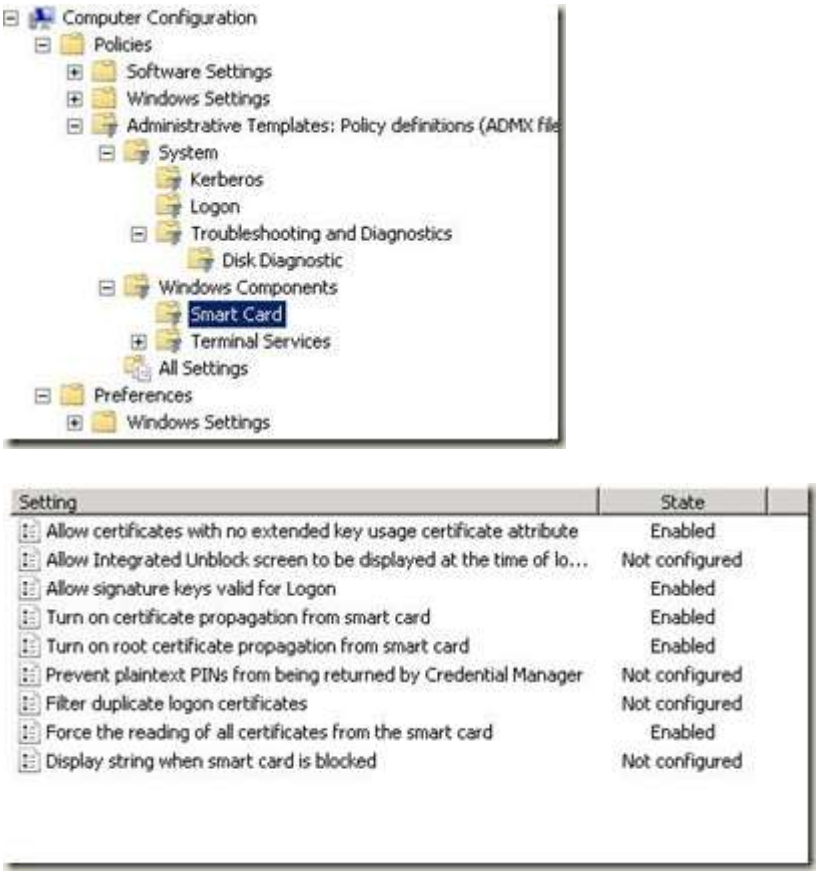


Figure 1

The important settings are:

HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\SmartCardCredentialProvider

AllowCertificatesWithNoEKU = 1

AllowSignatureOnlyKeys = 1

ForceReadingAllCertificates = 1

I then linked this policy to both the DCs and the client machines – see figure 2

For a real deployment you should link it at the domain level.

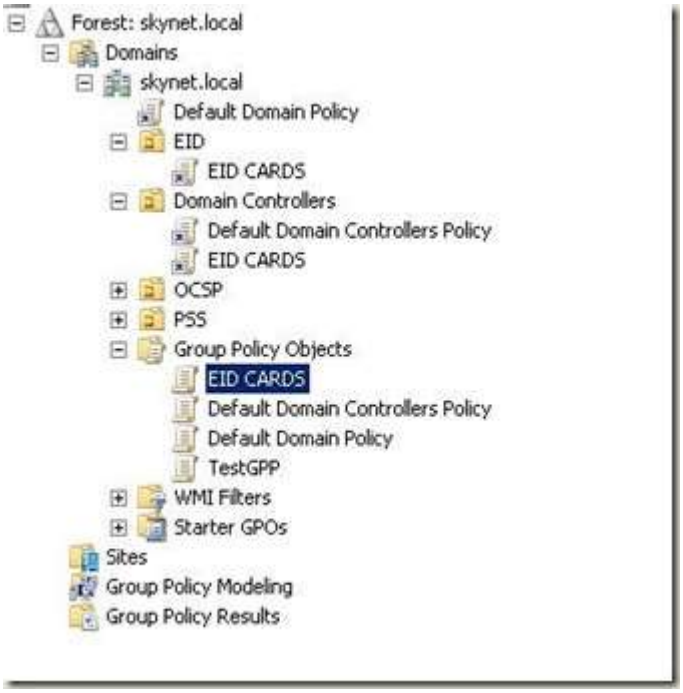


Figure 2

I also set the keys noted in http://support.microsoft.com/kb/887578 you need to set these on the KDC and on the client (note the different location depending on the client versus DC)

UseCachedCRLOnlyAndIgnoreRevocationUnknownErrors

CRLTimeoutPeriod

Please use common sense here – yes CRL checks will fail.

For smartcard logon, the client machine checks the DC cert, and the DC checks the smartcard client cert.

Due to the nature of the disjointed management, CRL locations etc.. we disabled any failures based off of CRL checks. I guess one can revoke ones citizenship… but I think you have bigger issues in your life if you had that happen.

(See the Appendix section for exact reg values etc.. exported directly from this config.)

There were 2 CA's in the cert path – a root and intermediate. However we could not reliably connect to any of them. We should think of them as dead. We do not control them nor can we dictate where or what they publish.

I imported the root cert into my ROOT store **on the Domain Controller and on the Vista clients**:

C:\temp\>certutil -addstore ROOT root1.cer

ROOT

Signature matches Public Key

Certificate "CN=TEST MyCountry Root CA, C=MY " added to store.

CertUtil: -addstore command completed successfully.

I imported the issuing into my intermediate store:

C:\temp\>certutil -addstore CA root2.cer

CA

Certificate "SERIALNUMBER=200501, CN=TEST Citizen CA, C=MY " added to store.

CertUtil: -addstore command completed successfully.

I imported the issuing CA into the NTAuth store as well. See http://support.microsoft.com/kb/281245 for more info on that.

At this point we should be good to go – except for one last item.

This was the subject on the cert:

Subject:

SERIALNUMBER=71717100052

G=SPAT 12345

SN=TEST

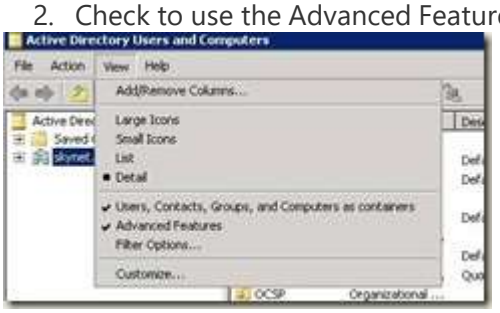
CN=SPAT TEST (Authentication)

C=MY

There was no SAN with a UPN – how were we to match the logon with a user?

You need to use cert mapping.

1. Open AD users and computers.
2. Check to use the Advanced Features.



3. Right click the user you want to map this card to and choose name mappings.

4. Now go ahead and choose the certificate you want to map to.

How did I get the cert? I exported the public portion (of course you can't export the private key data) via "certutil -scinfo"

You can also grab it from the local users cert store once cert propagation took place.

After you have imported the cert to the user you will see the following attribute populated:

Dn: CN=geert,OU=EID,DC=skynet,DC=local

accountExpires: 9223372036854775807 (never);

altSecurityIdentities: X509:<I>C=MY,CN=TEST Citizen CA,SERIALNUMBER=200501<S>C=MY ,CN=SPAT TEST(Authentication),SN=TEST,G=SPAT 12345,SERIALNUMBER= 71717100052;

badPasswordTime: 0 (never);

badPwdCount: 0;

cn: geert;

codePage: 0;

countryCode: 0;

displayName: geert;

distinguishedName: CN=geert,OU=EID,DC=skynet,DC=local;

dSCorePropagationData: 0x0 = ( );

givenName: geert;

instanceType: 0x4 = (WRITE);

lastLogoff: 0 (never);

lastLogon: 4/14/2008 7:10:22 PM Pacific Daylight Time;

logonCount: 13;

name: geert;

objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=skynet,DC=local; objectClass (4): top; person; organizationalPerson; ser;

objectGUID: 4964aa8e-e82b-4ca5-bf69-2a753917cde2;

objectSid: S-1-5-21-3046753738-575175152-1907242857-1110;

primaryGroupID: 513 = (GROUP_RID_USERS);

pwdLastSet: 4/4/2008 12:06:14 PM Pacific Daylight Time;

sAMAccountName: geert;

sAMAccountType: 805306368 = (NORMAL_USER_ACCOUNT);

userAccountControl: 0x200 = (NORMAL_ACCOUNT);

userPrincipalName: geert@skynet.local;

uSNChanged: 20635;

uSNCreated: 20629;

whenChanged: 4/4/2008 12:19:30 PM Pacific Daylight Time;

whenCreated: 4/4/2008 12:06:14 PM Pacific Daylight Time;

Once we set all these and rebooted – **we continued to fail the logon.** Sigh…

Time to bust out Mr debugger….I looked at what the Base CSP was calling for. Took a little while but here was the eventual failure code

0: kd> KL

ChildEBP RetAddr

WARNING: Stack unwind information not available. Following frames may be wrong.

01d6ebec 73b8b418 AZeBeIDMDRV!CardGetProperty

01d6ee78 73b8b603 basecsp!GetCachedCardProperty+0x138

01d6eea0 73b83889 basecsp!CspQueryKeySizes+0x42

01d6ef4c 73b81cbb basecsp!BuildSupportedAlgorithmsList+0x12a

01d6ef90 73b81bc6 basecsp!LocalGetProvParam+0x25c

01d6efd4 768e7723 basecsp!CPGetProvParam+0x42

01d6f024 75bf8586 ADVAPI32!CryptGetProvParam+0x4c

01d6f040 75be3213 kerberos!ScHelperGetProvParam+0x27

01d6f0a4 75be53df kerberos!__ScHelperGetSmartCardAlgorithms+0x5e

01d6f1e8 75bc5f13 kerberos!KerbBuildPkinitPreauthData+0x11e

01d6f270 75b9db74 kerberos!KerbBuildPreAuthData+0x8b

01d6f2f8 75b9cc97 kerberos!KerbGetPreAuthDataForRealm+0x13a

01d6f568 75b9d701 kerberos!KerbGetAuthenticationTicketEx+0x652

01d6f630 75babf8f kerberos!KerbGetTicketGrantingTicket+0x212

01d6f918 76161cfa kerberos!LsaApLogonUserEx2+0x1034

01d6f9e4 7613d34d LSASRV!NegLogonUserEx2+0x3cb

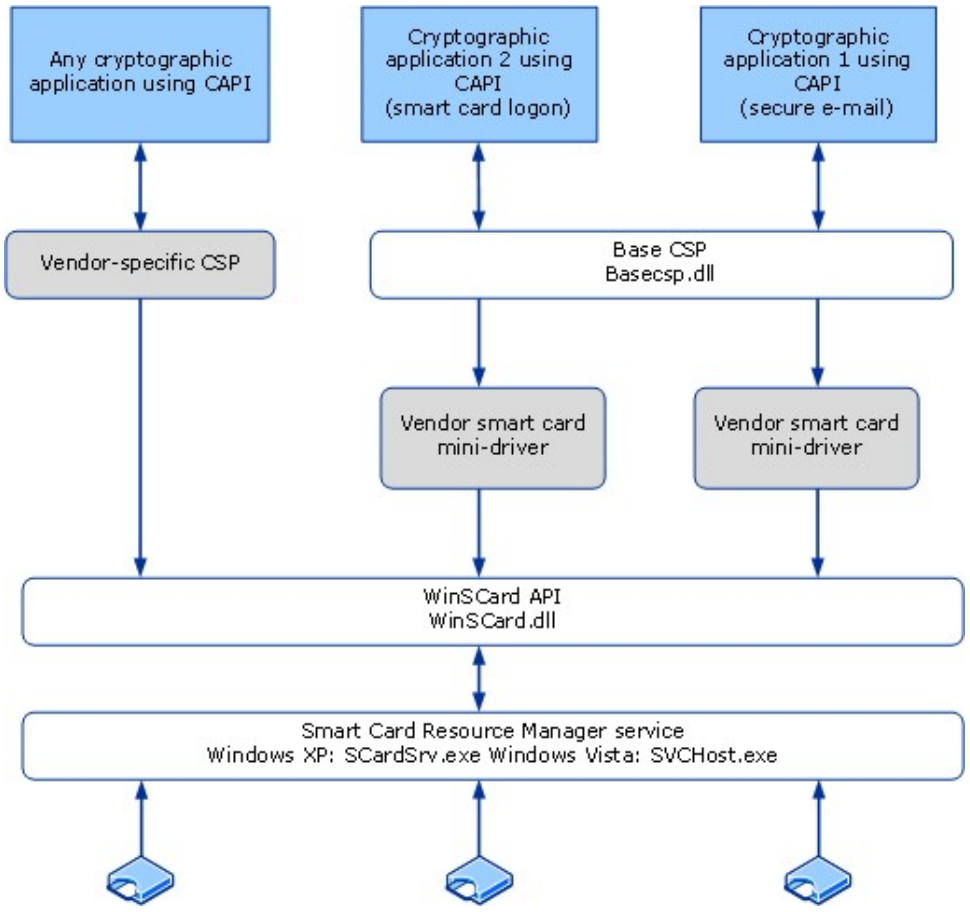
01d6fbc4 7613d071 LSASRV!LsapAuApiDispatchLogonUser+0x529

01d6fbd4 76165fd1 LSASRV!LpcLsaLogonUser+0x15

01d6fc94 76387ff9 LSASRV!DispatchAPIDirect+0x185

01d6fd64 7616289a Secur32!LsaLogonUser+0xe8

Here is a quick architectural diagram stolen from MSDN.. it shows you where the card module comes into play.



Smart card selection heuristics

When we call CardGetProperty we were specifying the following: **CP_CARD_KEYSIZES** and

dwFlags **AT_SIGNATURE** and then we would query for **AT_KEYEXCHANGE.**

We would succeed on the signature query but fail the exchange query.

Well , this particular identity card will only ever have AT_SIGNATURE, so the developer was returning SCARD_E_UNSUPPORTED_FEATURE when it was queried for AT_KEYEXCHANGE.

Once the card module returned SCARD_E_UNSUPPORTED_FEATURE – we translated that to Kerberos on the client as STATUS_SMARTCARD_SUBSYSTEM_FAILURE

Note:

We may have an issue here – I did look into it a little bit and from a card module perspective is certainly is OK to return that we don't support AT_KEYEXCHANGE.

However – the new policies also allow us to have a card which only has AT_SIGNATURE (in previous OS's the SC cert needed to have AT_KEYEXCHANGE see

http://support.microsoft.com/kb/281245)

"There are two predefined types of private keys. These keys are Signature Only(AT_SIGNATURE) and Key Exchange(AT_KEYEXCHANGE). Smartcard logon certificates must have a Key Exchange(AT_KEYEXCHANGE) private key type in order for smartcard logon to function correctly."

Up until now – we had not even gotten a chance to talk to the KDC.

Anyway, once we changed this to return supported we finally got around to calling the KDC and… again was **still failing the logon** . Sigh…

Here was the error:

Log Name: System

Source: Microsoft-Windows-Kerberos-Key-Distribution-Center

Date: 4/14/2008 3:07:26 PM

Event ID: 21

Task Category: None

Level: Warning

Keywords: Classic

User: N/A

Computer: 2k8entspat.skynet.local

Description:

The client certificate for the user SKYNET0\geert is not valid, and resulted in a failed smartcard logon. Please contact the user for more information about the certificate they're attempting to use for smartcard logon. The chain status was : **The certificate is not valid for the requested usage.**

In order to correct this I set the following registry key on the KDC.

http://support.microsoft.com/kb/936358

1. Click Start, click Run, type regedit, and then click OK.
2. Locate the following registry subkey:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kdc

3. Right-click Kdc, point to New, and then click DWORD Value.
4. Type **SCLogonEKUNotRequired**, and then press ENTER.
5. On the Edit menu, click Modify.
6. In the Value data box, type 1, and then click OK.
7. Exit Registry Editor.

Now – I can logon but it hangs at logging in for about 15 minutes before logon (at least it seems like 15 minutes)

Looking at the threads I can see that they are performing a number of LDAP and HTTP lookups for CRL's which they will never be able to get to.

I reset **CRLTimeoutPeriod** (under HKEY_Local_Machine\System\CurrentControlSet\Control\Lsa \Kerberos\Parameters\CRLTimeoutPeriod) to a value of 1 and now we can logon in a reasonable time.

APPENDIX

reg entries – please note that the "policies" key\values should be set via the GPO's discussed earlier.

Vista Client:

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters]

"UseCachedCRLOnlyAndIgnoreRevocationUnknownErrors"=dword:00000001

"CRLTimeoutPeriod"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\SmartCardCredentialProvider]

"AllowCertificatesWithNoEKU"=dword:00000001

"AllowSignatureOnlyKeys"=dword:00000001

"ForceReadingAllCertificates"=dword:00000001

Domain Controller Reg Entries:

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\kdc]

"UseCachedCRLOnlyAndIgnoreRevocationUnknownErrors"=dword:00000001

"SCLogonEKUNotRequired"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters]

"UseCachedCRLOnlyAndIgnoreRevocationUnknownErrors"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\SmartCardCredentialProvider]

"AllowCertificatesWithNoEKU"=dword:00000001

"AllowSignatureOnlyKeys"=dword:00000001

"ForceReadingAllCertificates"=dword:00000001

Various web resources..

http://www.microsoft.com/technet/security/guidance/identitymanagement/smrtcdcb/default.mspx -The Smart Card Deployment Cookbook

http://msdn2.microsoft.com/en-us/library/ms953432.aspx – The Smart Card Cryptographic Service Provider Cookbook

http://support.microsoft.com/kb/936358 – SCLogonEKUNotRequired

http://support.microsoft.com/kb/281245 – third party smartcard logon

http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/sc-minidriver_specs_V6-final.docx – card module specs

http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/sc-minidriver_certreqs.doc

http://msdn2.microsoft.com/en-us/library/bb905527.aspx – Windows Vista Smart Card Infrastructure

Certificate from ID card as a reference to what we were trying to get to logon:

X509 Certificate:

Version: 3

Serial Number: 01000000000115d1d292a9

Signature Algorithm:

Algorithm ObjectId: 1.2.840.113549.1.1.5 sha1RSA

Algorithm Parameters:

05 00

Issuer:

SERIALNUMBER=200501

CN= TEST Citizen CA

C=MY

NotBefore: 10/24/2007 4:42 AM

NotAfter: 10/24/2009 4:42 AM

Subject:

SERIALNUMBER=71717100052

G=SPAT 12345

SN=TEST

CN=SPAT TEST (Authentication)

C=MY

Public Key Algorithm:

Algorithm ObjectId: 1.2.840.113549.1.1.1 RSA (RSA_SIGN)

Algorithm Parameters:

05 00

Public Key Length: 1024 bits

Public Key: UnusedBits = 0

0000 30 81 89 02 81 81 00 81 38 f4 f7 c8 d7 69 ce c5

0010 08 8d b1 42 a6 88 aa 8d 3f 0c 5b a0 6e 44 a1 b7

0020 df 85 fc fe 42 fc 00 d0 d8 01 88 22 32 47 12 68

0030 e1 c3 34 d3 30 70 87 ac 43 b1 d4 b3 f1 26 6e e3

0040 dd 62 6c 67 b6 0d ff ec c4 b0 d2 b8 fa da 9c 6e

0050 e9 9c b0 8b cf ba bf 8d 1f 74 89 48 d1 e0 6f 9b

0060 ab bb e5 53 99 19 d0 58 fb ed 3c f1 c3 4c a2 2b

0070 c9 8d 13 41 0d 7e cf 95 b1 31 80 bd 4e 38 1b fb

0080 e3 11 49 c1 55 e5 31 02 03 01 00 01

Certificate Extensions: 6

2.5.29.35: Flags = 0, Length = 18

Authority Key Identifier

KeyID=d6 a5 fe 65 26 bf 28 6c 16 15 d7 fa 7e 3d da 9f a9 ee 7d 1d

1.3.6.1.5.5.7.1.1: Flags = 0, Length = 73

Authority Information Access

[1]Authority Info Access

Access Method=Certification Authority Issuer (1.3.6.1.5.5.7.48.2)

Alternative Name:

URL=http://certs.test-eid.mycountry.eu/mycountry.crt

[2]Authority Info Access

Access Method=On-line Certificate Status Protocol (1.3.6.1.5.5.7.48.1)

Alternative Name:

URL=http://ocsp.test-eid.mycountry.eu

2.5.29.32: Flags = 0, Length = 48

Certificate Policies

[1]Certificate Policy:

Policy Identifier=0.3.2062.7.1.1.401.1

[1,1]Policy Qualifier Info:

Policy Qualifier Id=CPS

Qualifier:

http://repository.test-eid.mycountry.eu

2.5.29.31: Flags = 0, Length = 3b

CRL Distribution Points

[1]CRL Distribution Point

Distribution Point Name:

Full Name:

URL=http://crl.test-eid.mycountry.eu/eidc200501.crl

2.5.29.15: Flags = 1(Critical), Length = 4

Key Usage

Digital Signature (80)

2.16.840.1.113730.1.1: Flags = 0, Length = 4

Netscape Cert Type

SSL Client Authentication, SMIME (a0)

Signature Algorithm:

Algorithm ObjectId: 1.2.840.113549.1.1.5 sha1RSA

Algorithm Parameters:

05 00

CardGetProperty

Description:
The function is modeled after the query functions of CAPI for keys. It takes a LPWSTR indicating which parameter is being requested. It then returns data written into pbData:

```
DWORD WINAPI CardGetProperty(
    IN   PCARD_DATA      pCardData,
    IN   LPWSTR          wszProperty,
    OUT  PBYTE           pbData,
    IN   DWORD           cbData,
    OUT  PDWORD          pdwDataLen,
    IN   DWORD           dwFlags
);
```

Input:

pCardData Address of CARD_DATA structure.
wszProperty LPWSTR indicating which property is requested.
pdData Byte pointer to data buffer to receive the data.
cbData Length of input buffer.
pdwDataLen Pointer to a DWORD receiving the actual data length returned.
dwFlags Flags.

Output:

Return value Zero on success; nonzero on failure.

Comments:

- CardGetProperty should check the dwFlags value. Unless dwFlags is specified for the property and the value is nonzero, it should fail and return SCARD_E_INVALID_PARAMETER.
- If an unsupported wszProperty is passed to CardGetProperty, it should fail and return SCARD_E_INVALID_PARAMETER. Implementing all of the following properties is mandatory unless explicitly stated otherwise. Any minidriver can choose to define and support optional custom properties that are not defined in this specification.
- If cbData is less than the length of the buffer to be returned, CardGetProperty should return ERROR_INSUFFICIENT_BUFFER.

Important note:
Careful attention must be taken when returning CP_READ_ONLY_CARD as true. When this property is returned as true, all write operations to the card are blocked at the BaseCSP layer.

- The format of pbData is different depending on the wszProperty parameter that is passed to the function. The following table is a list of the different types that pbData takes depending on wszProperty (the structures are serialized as byte arrays).

wszProperty	pbData type	pbData value
CP_CARD_FREE_SPACE		`typedef struct _CARD_FREE_SPACE_INFO { DWORD dwVersion; DWORD dwBytesAvailable; DWORD dwKeyContainersAvailable; DWORD dwMaxKeyContainers; } CARD_FREE_SPACE_INFO, *PCARD_FREE_SPACE_INFO;`
CP_CARD_CAPABILITIES		`typedef struct _CARD_CAPABILITIES { DWORD dwVersion; BOOL fCertificateCompression; BOOL fKeyGen; } CARD_CAPABILITIES, *PCARD_CAPABILITIES;`
CP_CARD_KEYSIZES		dwFlags indicates key type to be queried. This is one of the AT_* defined values, for example, AT_SIGNATURE or AT_ECDSA_P256. `typedef struct _CARD_KEY_SIZES { DWORD dwVersion; DWORD dwMinimumBitlen; DWORD dwDefaultBitlen; DWORD dwMaximumBitlen; DWORD dwIncrementalBitlen; } CARD_KEY_SIZES, *PCARD_KEY_SIZES;`
CP_CARD_READ_ONLY	BOOL	If True, all write operations are blocked at the CSP layer. This flag also affects the data cache. If the card indicates that it is read only, the BaseCSP/KSP does not write to the cardcf file.

CP_CARD_CACHE_MODE	DWORD	`#define CP_CACHE_MODE_GLOBAL_CACHE 1`
		`#define CP_CACHE_MODE_SESSION_ONLY 2`
		`#define CP_CACHE_MODE_NO_CACHE 3`

CP_SUPPORTS_WIN_X509_ENROLLMENT	BOOL	Indicates whether Windows PKI should be allowed to write or renew certificates on the card. This should be used to avoid unexpected results due to lack of support for multiple PINs in Windows PKI enrollment client.
CP_CARD_GUID	BYTE[]	In this case, pbData is a buffer containing a unique GUID for the card. This value must exactly match the GUID contained in the “cardid” file.
CP_CARD_SERIAL_NO	BYTE[]	In this case, pbData is a buffer containing a serial number for the card. The format of the serial number is opaque to the BaseCSP and is intended for other applications that query the card minidriver directly. This is an optional property that may or may not be supported by the card.
CP_CARD_PIN_INFO	PIN_INFO	In this case, pbData is a PIN_INFO structure containing information about the PIN. The dwFlags parameter contains the identifier of the PIN to return.
CP_CARD_LIST_PINS	PIN_SET	In this case, pbData contains a PIN_SET indicating by a bit-mask what entities the card currently uses.
CP_CARD_AUTHENTICATED_STATE	PIN_SET	In this case, pbData contains a PIN_SET indicating by a bit-mask what entities the card currently authenticates. This is an optional property that may or may not be supported by the card.
CP_CARD_PIN_STRENGTH_VERIFY		In this case, pbData contains a bit mask of one or more of the following values: • CARD_PIN_STRENGTH_PLAINTEXT – Card can accept a plaintext PIN for authentication. • CARD_PIN_STRENGTH_SESSION_PIN – Card can generate a session PIN that can be used for subsequent authentications. • The dwFlags parameter contains the identifier of the PIN to return.

		The following points apply to PIN strength: • Currently the PIN strength is ignored for EmptyPinType and ChallengeResponsePinType. • Even if CARD_PIN_STRENGTH_SESSION_PIN is set for a PIN, the plaintext PIN must also be accepted for authentication. This is because trusted processes in Windows may use the plaintext PIN.

From <https://blogs.msdn.microsoft.com/spatdsg/2008/04/17/smartcard-in-2008-and-vista-national-id-card-no-upn-no-eku-no-problem/>