Cryptonite

Áron Szabó
Péter Psenák

H.A.C.K.
Hackerspace Budapest (hsbp.org)
2013. december 13.

Backgrounds

„There's no place to hide
When two worlds collide"

X.509 vs. non-X.509

- PGP, GPG, DNSSEC...
- SSL/TLS, XML signature, S/MIME, CMS, code signing...

Web of Trust vs. Trusted Third Party

- PGP, GPG...
- CA-hierarchy

Backgrounds

Diffie-Hellman

```
Diffie-Hellman

prime number:
                    p = 23                          (public)

base number:
                    g = 5                           (public)

secret numbers:
                    a = 6                           (private)
                    b = 15                          (private)

public values:
                    A = g^a mod p =  5^6  mod 23 = 8
                    B = g^b mod p =  5^15 mod 23 = 19

shared secret key:
                    Z = A^b mod p =  8^15 mod 23 = 2
                    Z = B^a mod p = 19^6  mod 23 = 2
```

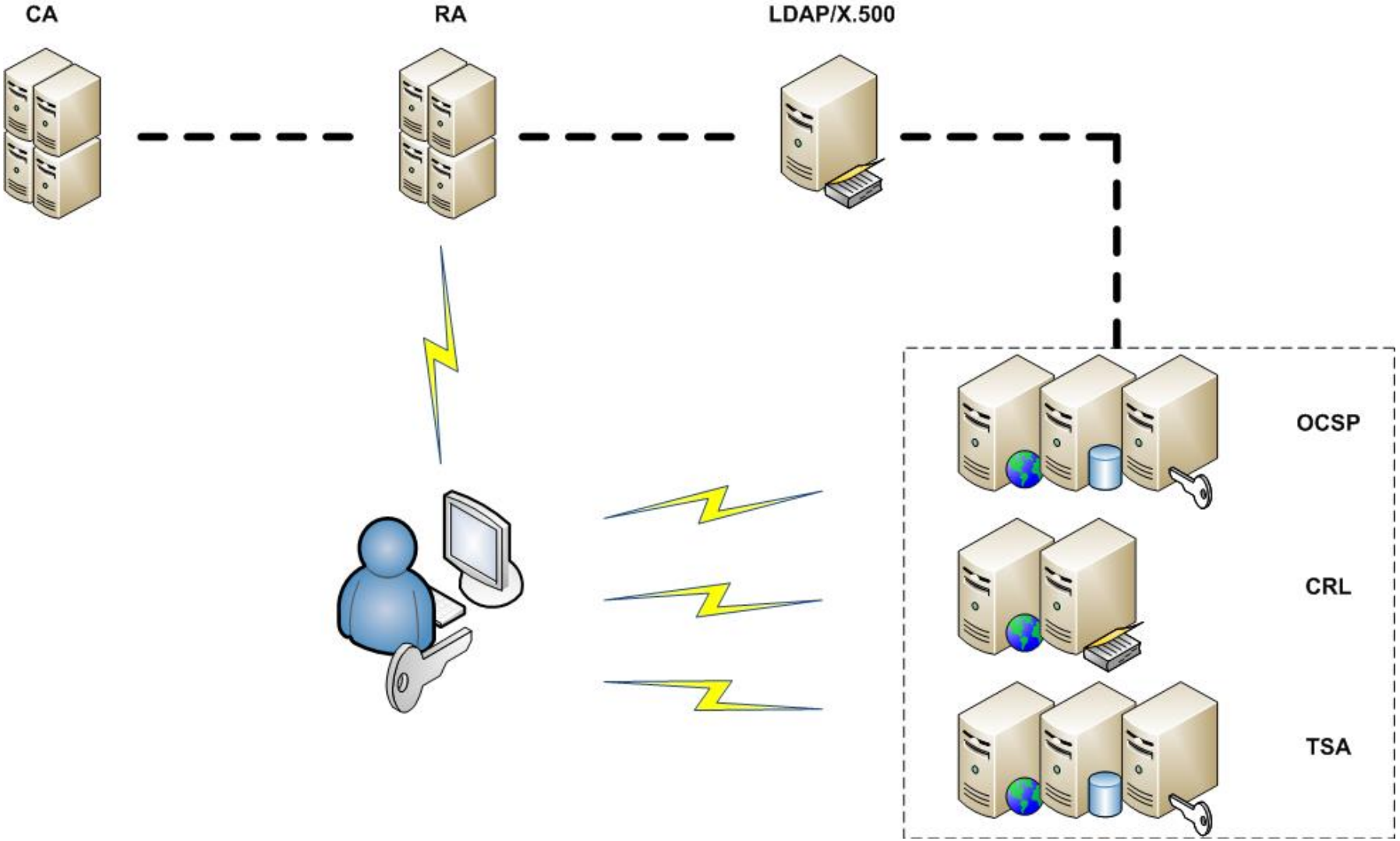
Backgrounds

RSA

```
RSA

prime numbers and φ(n):
                        p = 13                          (private)
                        q = 11                          (private)

                        n    = p * q = 143              (public)
                        φ(n) = (p - 1) * (q - 1) = 120

coprime number to φ(n):
                        e = 7                           (public)
                        e * d   ≡ 1 (mod φ(n))
                        7 * 103 = 721 = 1 + 6 * 120
                        d = 103                         (private)

public key:
                        n = 143
                        e = 7

private key:
                        n = 143
                        d = 103

operations:
                        data      = 3

                        signature =      (data)^d mod n =  3^103  mod 143 = 16
                        data      = (signature)^e mod n = 16^7   mod 143 = 3

                        cipher    =      (data)^e mod n =  3^7    mod 143 = 42
                        data      =    (cipher)^d mod n = 42^103  mod 143 = 3
```

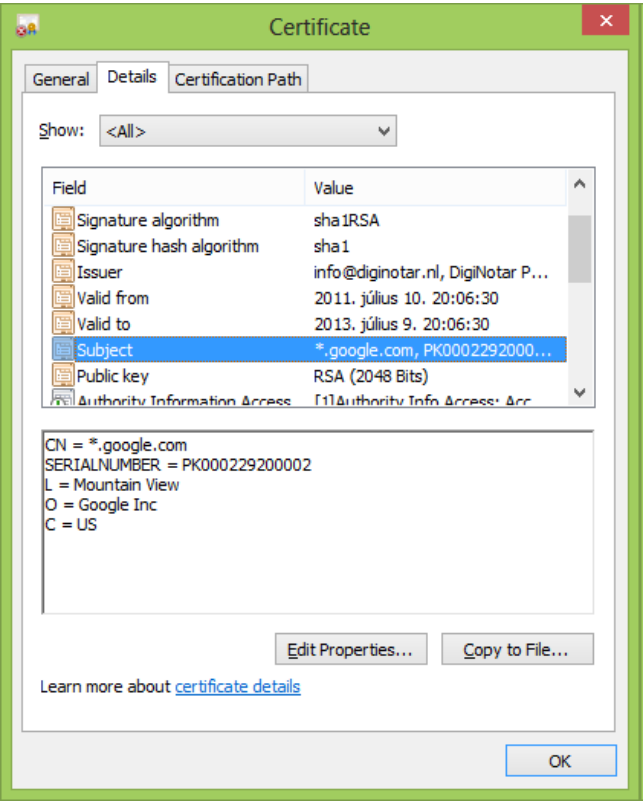
CA anatomy

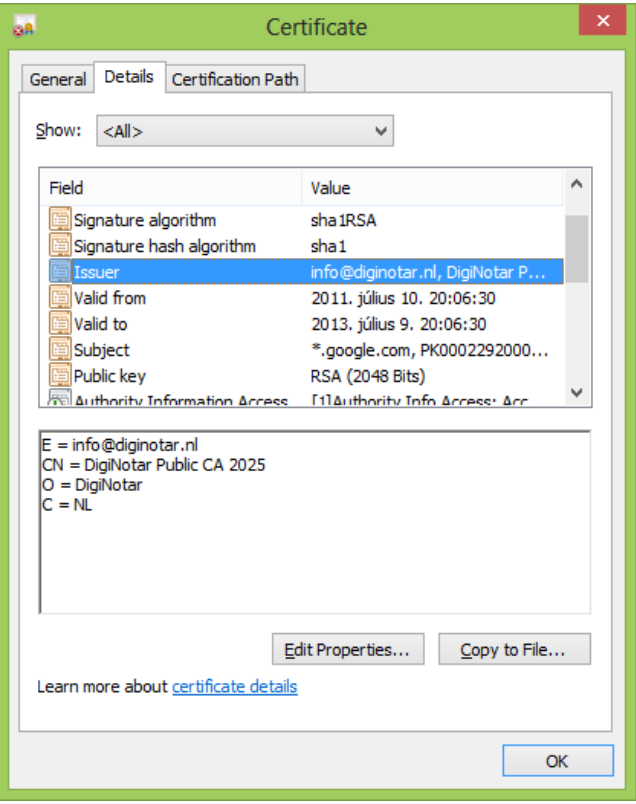
architecture

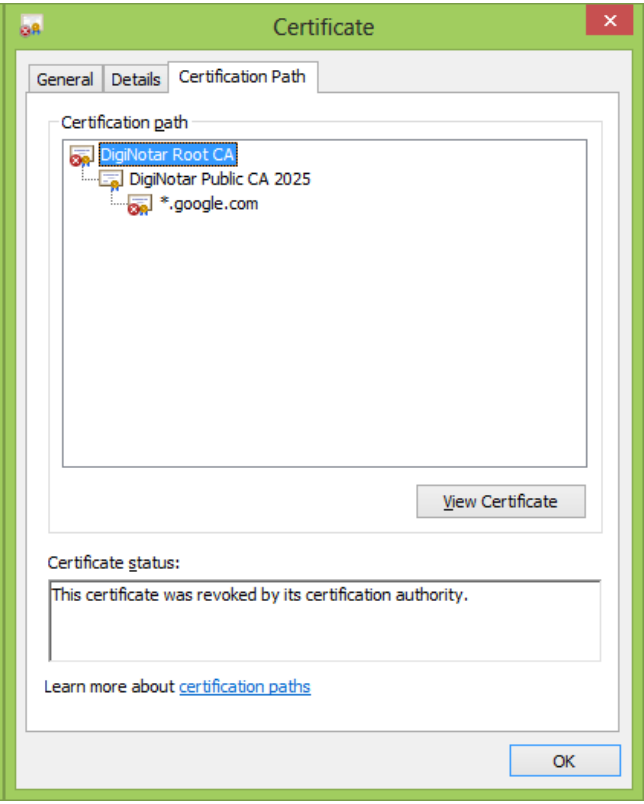


CA anatomy

architecture and DigiNotar







CA anatomy

architecture and DigiNotar

CA server logging

- "CA servers did not log to a separate secure log server. All the investigated log files originated from servers that had been compromised"
- "integrity of blocks of data within the log files can be verified using a signature [...] Two log files failed the verification by the CA software"

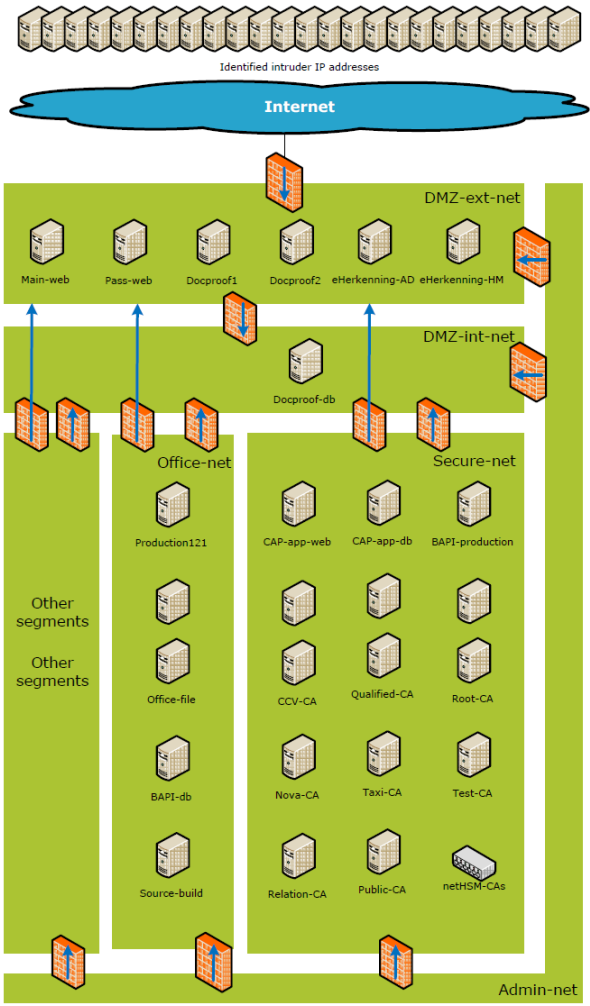
CA software

- "The product used was the RSA Certificate Manager (RSA CM). [...] Older versions of this software are known as RSA Keon."
- "no link could be established between a certificate and an entry in the log files on the basis of a serial number."

CA keystore

- "DigiNotar used nCipher netHSM 500s"
- "CA servers had access to the nCipher netHSM that was also located in Secure-net"
- "on the Qualified-CA server, it appeared that some of the DLLs that were used to access the netHSM had been modified"
- "private keys were activated in the netHSM using smartcards"
- "log entries were found that indicated the automatic generation of a Certificate Revocation List (CRL) [...] which can only occur if a private key was active on the netHSM"

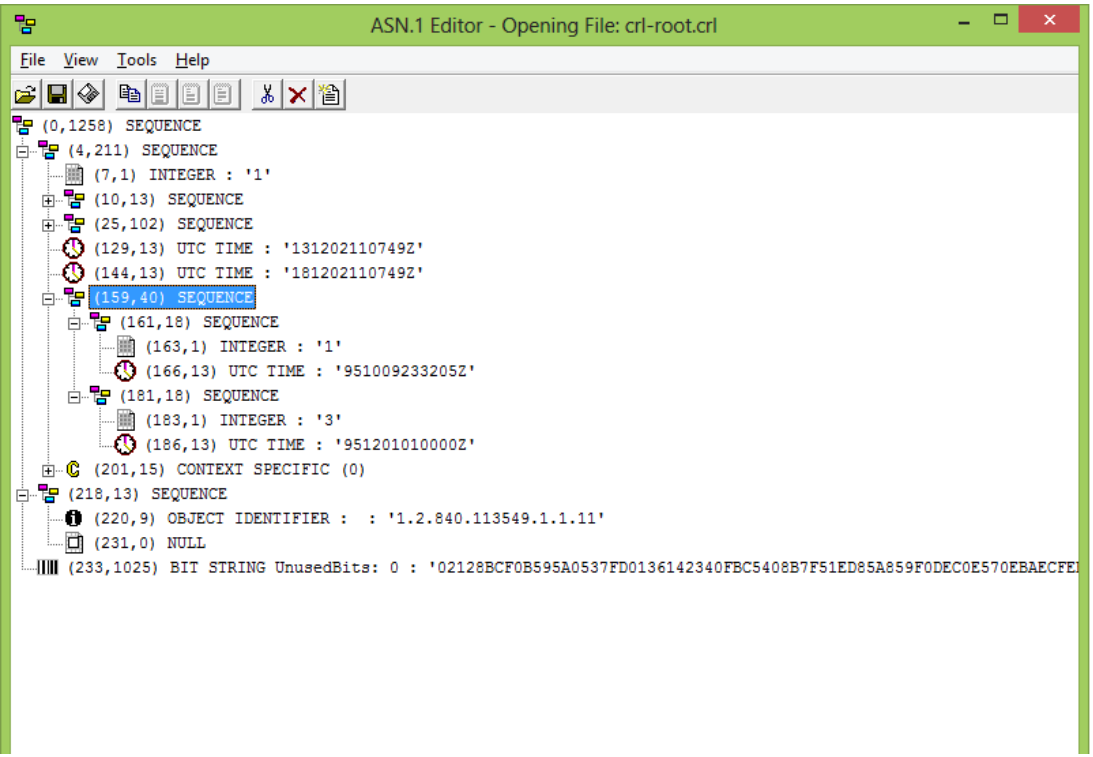
Date	Notes
2011-06-17	"the Main-web and the Docproof2 web server were compromised"
2011-07-01	"first scanning activity occurred in Secure-net"
2011-07-10	"first rogue certificate was successfully created on the Relation-CA server [...]OCSP requests for rogue certificates started arriving [...] from an DSL subscriber in Iran"
2011-07-27	"First OCSP request at DigiNotar for the rogue wildcard Google certificate"
2011-07-28	"attempts were made to verify the rogue login.yahoo.com certificate by IP addresses originating from the Islamic Republic of Iran"
2011-08-04	"massive activity on the OCSP responder for a rogue *.google.com certificate originating from the Islamic Republic of Iran"
2011-08-31	"Google Chrome blacklisted a list of known rogue serial numbers"



CA anatomy

CRL caching, OCSP stapling and legal requirements

- signed data
- issued periodically (e.g. 4 hours, 24 hours, in case of certificate revocation)

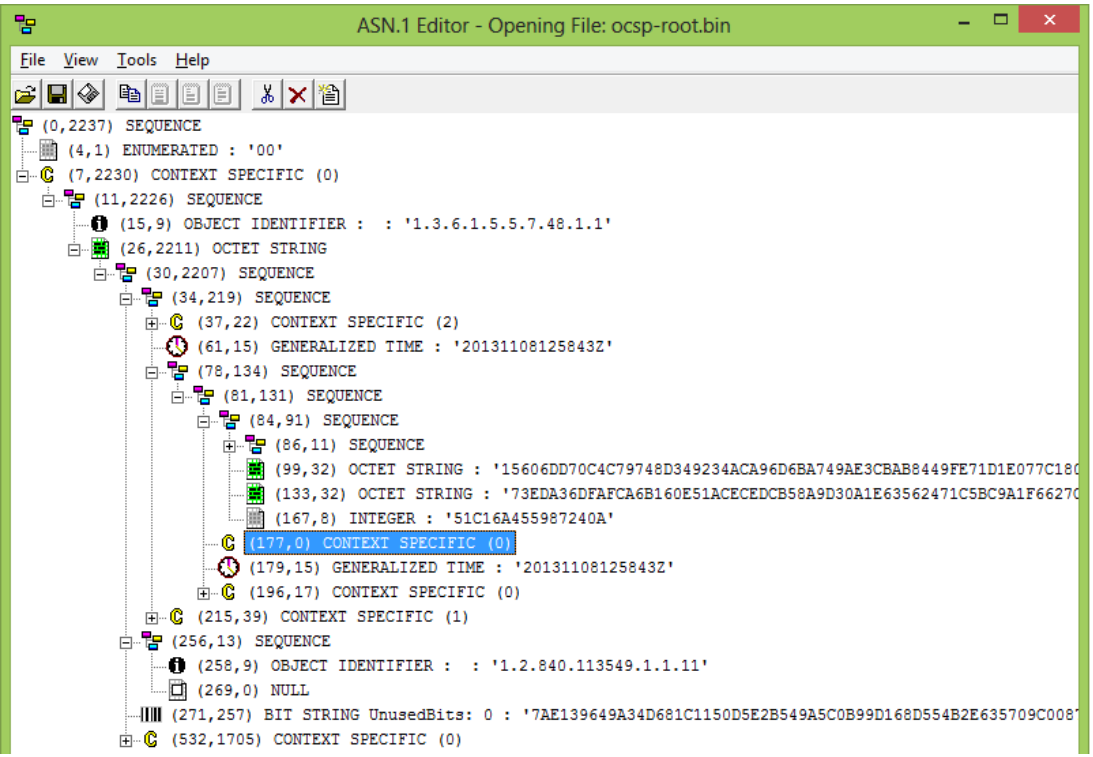



```
CertificateList

   CertificateList ::= SEQUENCE {
      tbsCertList                   TBSCertList,
      signatureAlgorithm            AlgorithmIdentifier,
      signatureValue                BIT STRING
                                 }

   TBSCertList     ::= SEQUENCE {
      version                       Version OPTIONAL,
      signature                     AlgorithmIdentifier,
      issuer                        Name,
      thisUpdate                    Time,
      nextUpdate                    Time OPTIONAL,
      revokedCertificates           SEQUENCE OF SEQUENCE {
         userCertificate                                  CertificateSerialNumber,
         revocationDate                                   Time,
         crlEntryExtensions                               Extensions OPTIONAL
                                                         } OPTIONAL,
      crlExtensions              [0] EXPLICIT Extensions OPTIONAL
                                 }
```

CA anatomy

CRL caching, OCSP stapling and legal requirements

- signed data
- instant response based on actual state of CA database




```
BasicOCSPResponse

   BasicOCSPResponse ::= SEQUENCE {
      tbsResponseData                 ResponseData,
      signatureAlgorithm              AlgorithmIdentifier,
      signature                       BIT STRING,
      certs                         [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
                                    }

   ResponseData      ::= SEQUENCE {
      version                       [0] EXPLICIT Version DEFAULT v1,
      responderID                     ResponderID,
      producedAt                      GeneralizedTime,
      responses                       SEQUENCE OF SingleResponse,
      responseExtensions            [1] EXPLICIT Extensions OPTIONAL
                                    }

   SingleResponse    ::= SEQUENCE {
      certID                          CertID,
      certStatus                      CertStatus,
      thisUpdate                      GeneralizedTime,
      nextUpdate                    [0] EXPLICIT GeneralizedTime OPTIONAL,
      singleExtensions              [1] EXPLICIT Extensions OPTIONAL
                                    }

   CertStatus        ::= CHOICE    {
      good                          [0] IMPLICIT NULL,
      revoked                       [1] IMPLICIT RevokedInfo,
      unknown                       [2] IMPLICIT UnknownInfo
                                    }

   RevokedInfo       ::= SEQUENCE {
      revocationTime                  GeneralizedTime,
      revocationReason              [0] EXPLICIT CRLReason OPTIONAL
                                    }

   UnknownInfo       ::= NULL
```

CA anatomy

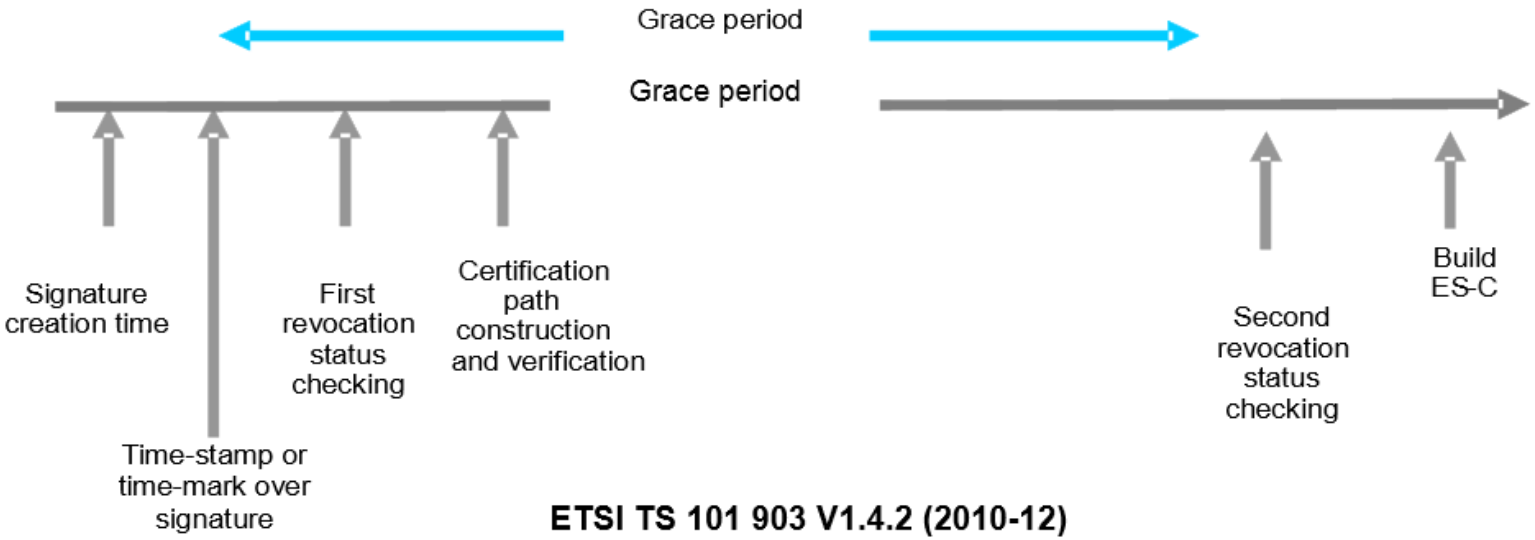
CRL caching, OCSP stapling and legal requirements

- good idea for short-term signatures (e.g. SSL/TLS)
- can cause conflicts with legal requirements

RFC 6066 TLS Extension Definitions January 2011

8. Certificate Status Request

Constrained clients may wish to use a certificate-status protocol such as OCSP [RFC2560] to check the validity of server certificates, in order to avoid transmission of CRLs and therefore save bandwidth on constrained networks. This extension allows for such information to be sent in the TLS handshake, saving roundtrips and resources.

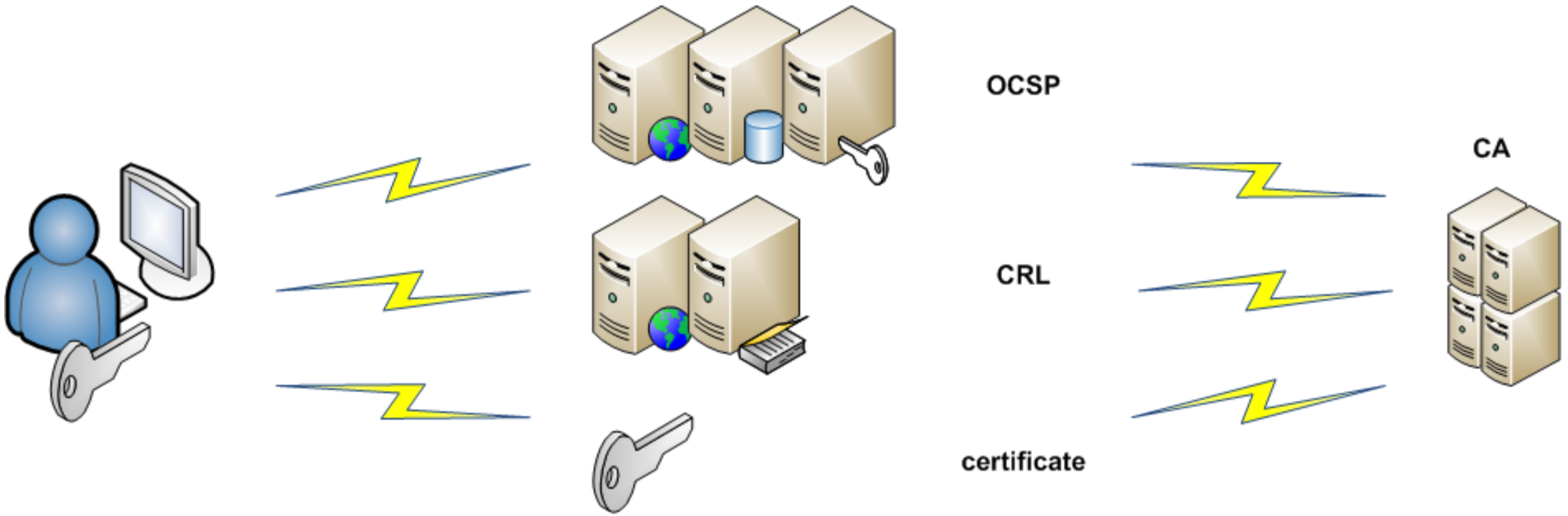


ETSI TS 101 903 V1.4.2 (2010-12)

CA anatomy

problem of different root CA entities

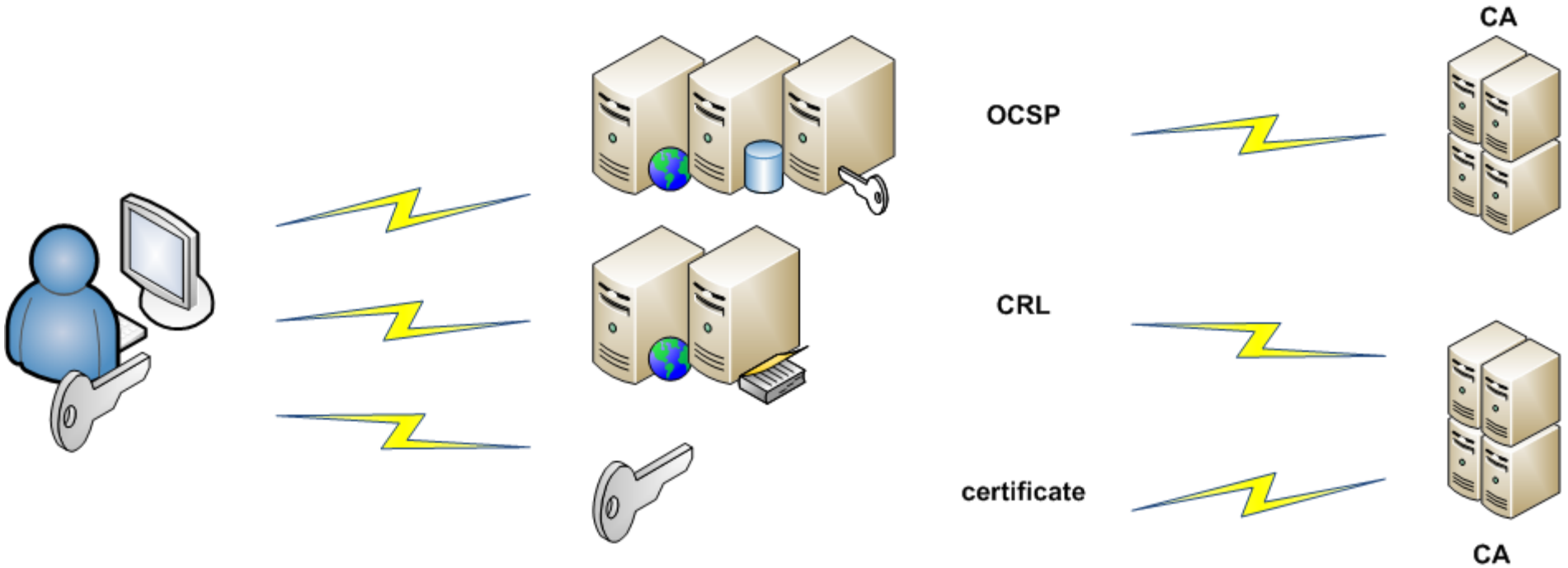
- both CA entities are trusted
- commonName strings are similar
- signature verification application still does not know the backend system



CA anatomy

problem of different root CA entities

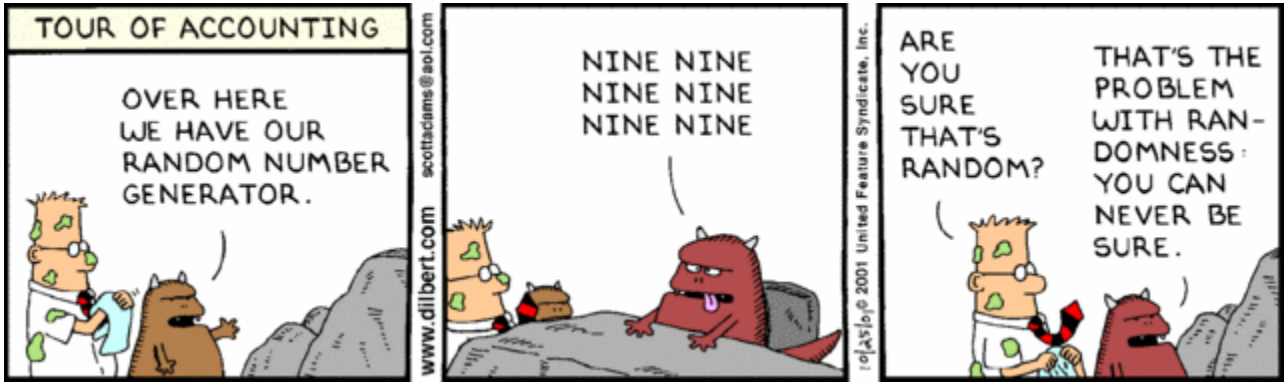
- both CA entities are trusted
- commonName strings are similar
- signature verification application still does not know the backend system



CA anatomy

code review and OpenSSL

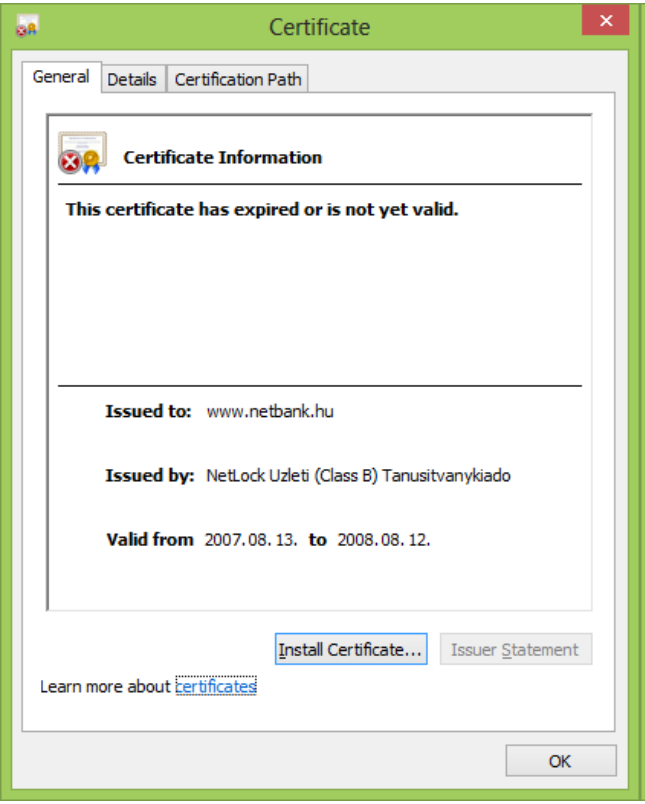
- OpenSSL: 2006-09-17 – 2008-05-13 (0.9.8c-1 – 0.9.8g-9)
- ssleay_rand_add() and process ID (PID): N * 32768 key pairs
- affected Hungarian sites (trusted CAs of Microsoft Windows)
- who generated the vulnerable key pairs and the PKCS#10?

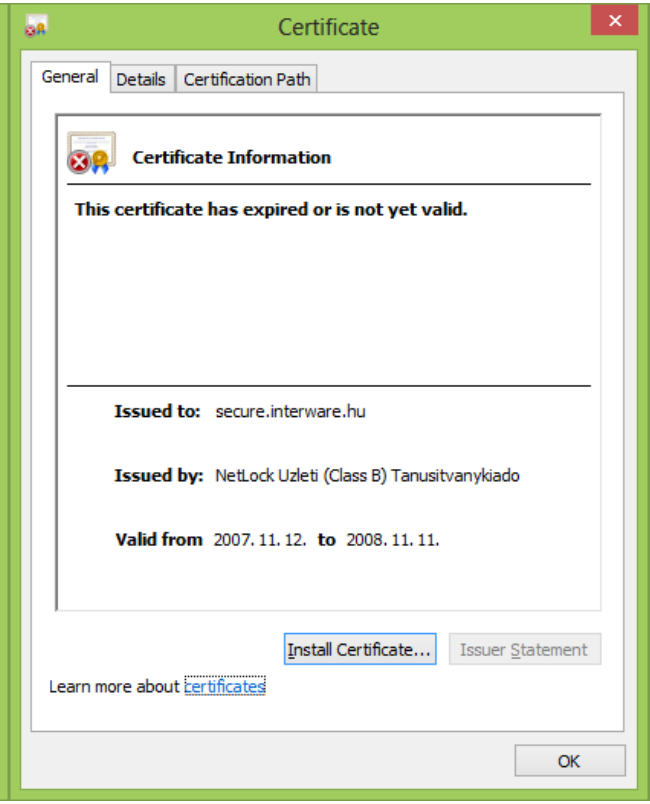


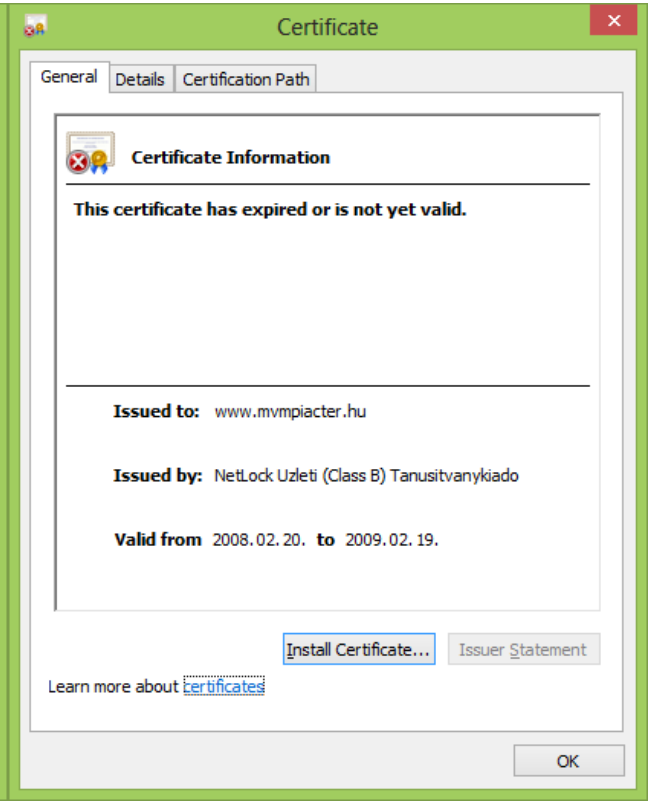
CA anatomy

code review and OpenSSL

- affected Hungarian sites (trusted Hungarian CA)
- bank (MagNet), server hosting and ISP (GTS), critical infrastructure (MVM)



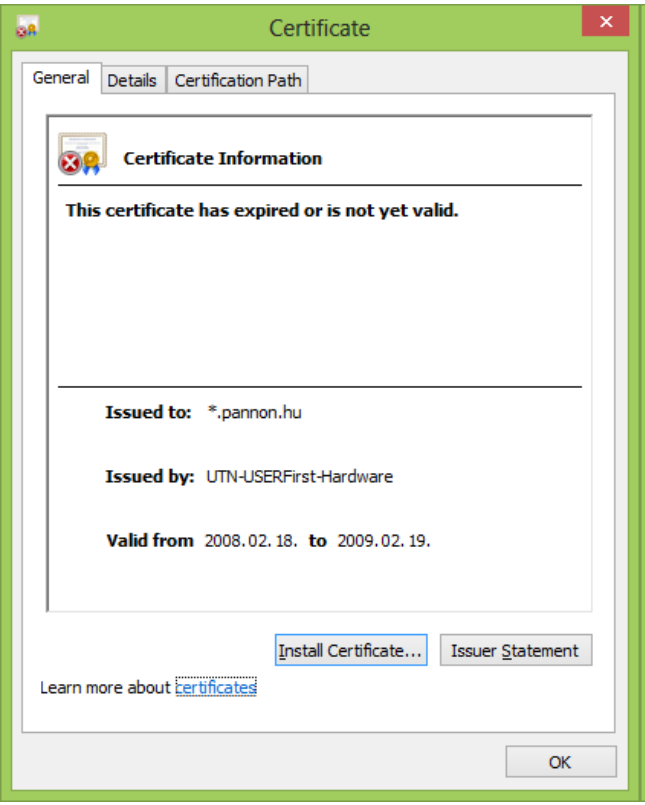


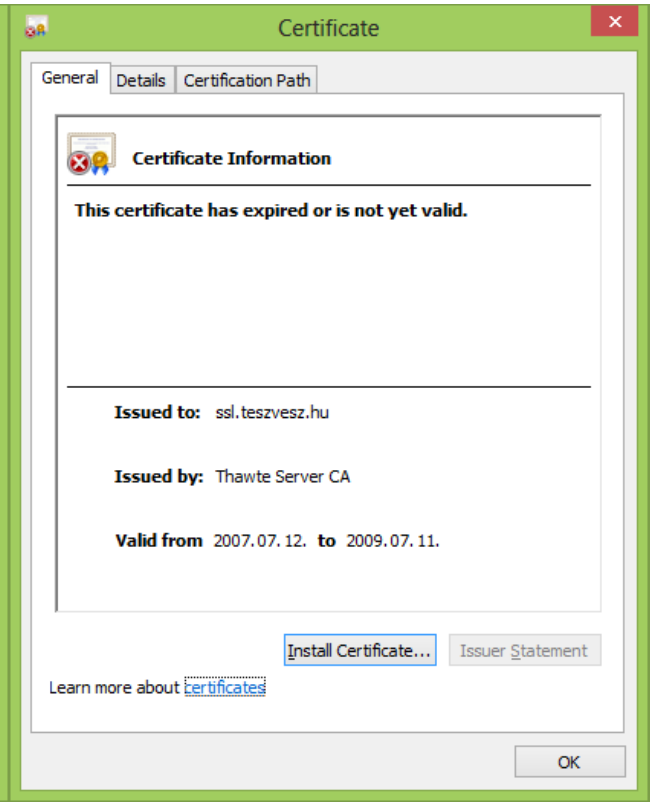


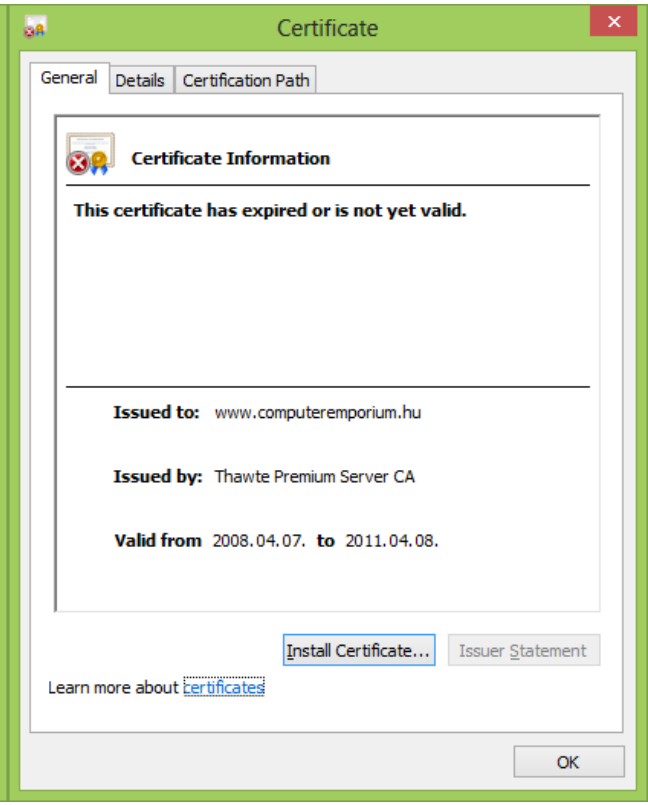
CA anatomy

code review and OpenSSL

- affected Hungarian sites (other trusted CAs)
- mobile operator (Telenor), auction (TeszVesz), webshop (ComputerEmporium)

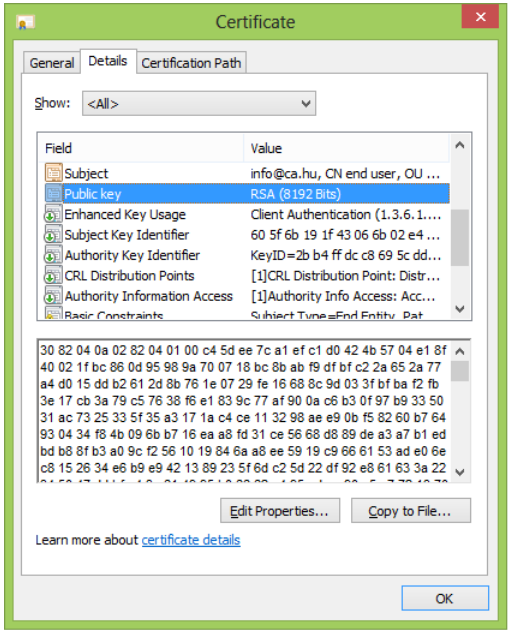




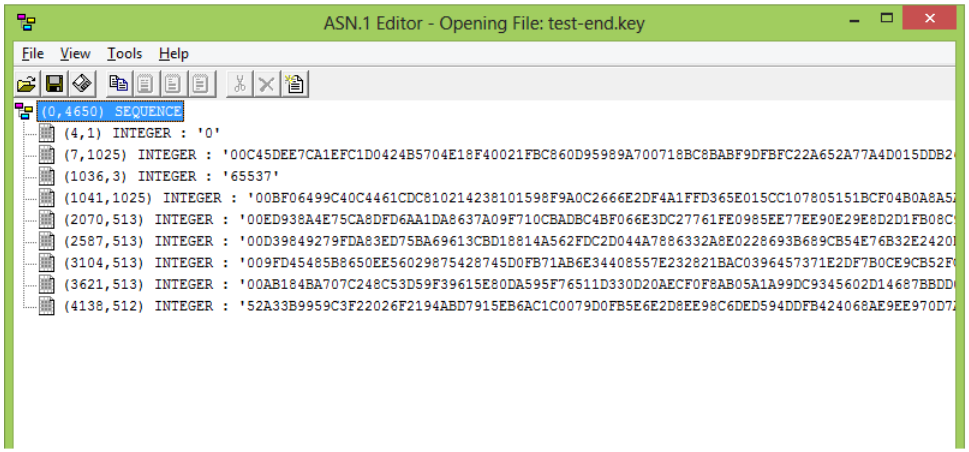


X.509 anatomy

RSAPublicKey, RSAPrivateKey

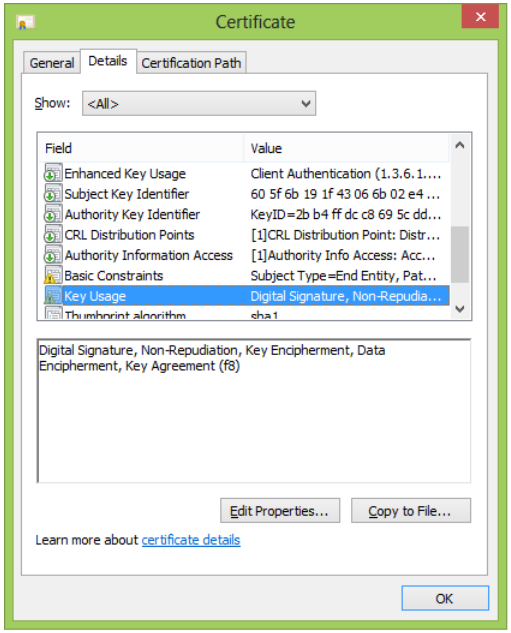



```
RSAPublicKey

   RSAPublicKey  ::= SEQUENCE {
                       modulus           INTEGER,  -- n
                       publicExponent    INTEGER   -- e
                     }

RSAPrivateKey

   RSAPrivateKey ::= SEQUENCE {
                       version           Version,
                       modulus           INTEGER,  -- n
                       publicExponent    INTEGER,  -- e
                       privateExponent   INTEGER,  -- d
                       prime1            INTEGER,  -- p
                       prime2            INTEGER,  -- q
                       exponent1         INTEGER,  -- d mod (p-1)
                       exponent2         INTEGER,  -- d mod (q-1)
                       coefficient       INTEGER,  -- (inverse of q) mod p
                       otherPrimeInfos   OtherPrimeInfos OPTIONAL
                     }
```

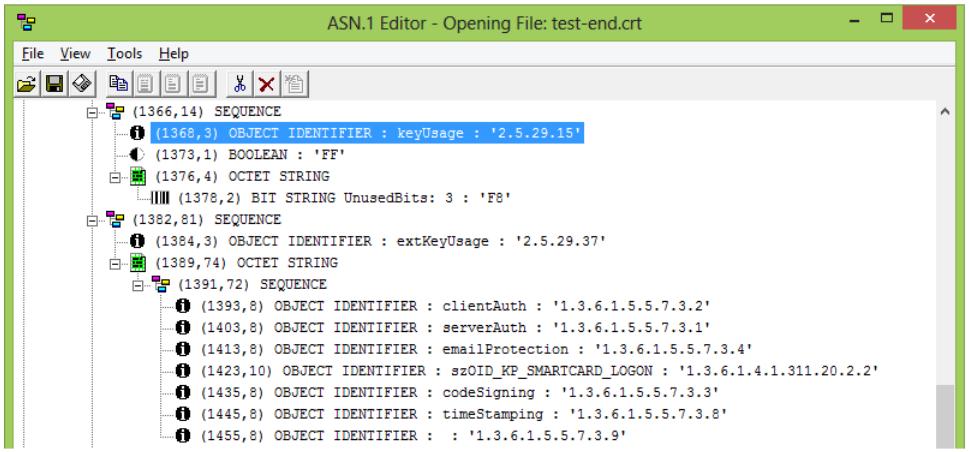


X.509 anatomy

keyUsage

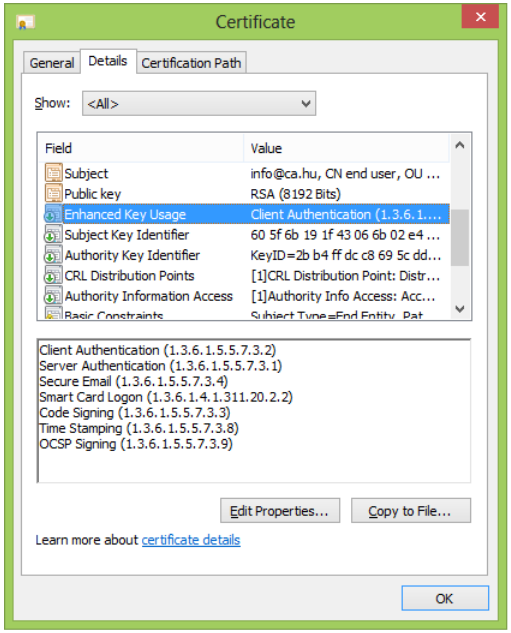



```
keyUsage

   id-ce-keyUsage OBJECT IDENTIFIER ::= { id-ce 15 }

   KeyUsage ::= BIT STRING {
                                digitalSignature  (0),
                                nonRepudiation    (1),
                                keyEncipherment   (2),
                                dataEncipherment  (3),
                                keyAgreement      (4),
                                keyCertSign       (5),
                                cRLSign           (6),
                                encipherOnly      (7),
                                decipherOnly      (8)
                               }
```

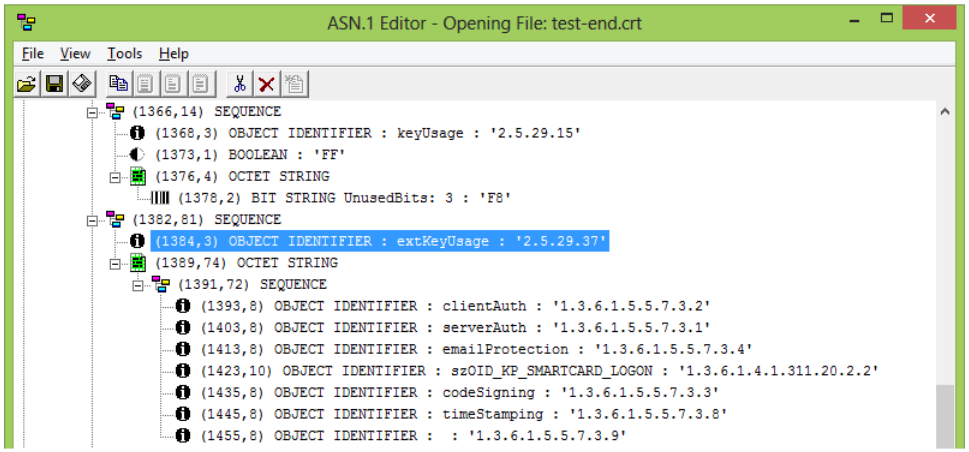


X.509 anatomy

extKeyUsage

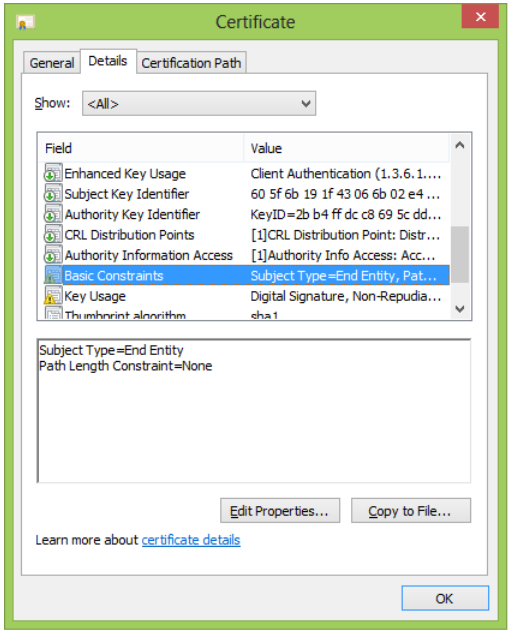



```
extKeyUsage

   id-ce-extKeyUsage OBJECT IDENTIFIER ::= { id-ce 37 }

   ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId

   KeyPurposeId ::= OBJECT IDENTIFIER

   anyExtendedKeyUsage   OBJECT IDENTIFIER ::= { id-ce-extKeyUsage 0 }

   id-kp                 OBJECT IDENTIFIER ::= { id-pkix 3 }

   id-kp-serverAuth      OBJECT IDENTIFIER ::= { id-kp 1 }
                         -- TLS WWW server authentication
                         -- Key usage bits that may be consistent: digitalSignature,
                         -- keyEncipherment or keyAgreement

   id-kp-clientAuth      OBJECT IDENTIFIER ::= { id-kp 2 }
                         -- TLS WWW client authentication
                         -- Key usage bits that may be consistent: digitalSignature
                         -- and/or keyAgreement

   id-kp-codeSigning     OBJECT IDENTIFIER ::= { id-kp 3 }
                         -- Signing of downloadable executable code
                         -- Key usage bits that may be consistent: digitalSignature

   id-kp-emailProtection OBJECT IDENTIFIER ::= { id-kp 4 }
                         -- Email protection
                         -- Key usage bits that may be consistent: digitalSignature,
                         -- nonRepudiation, and/or (keyEncipherment or keyAgreement)

   id-kp-timeStamping    OBJECT IDENTIFIER ::= { id-kp 8 }
                         -- Binding the hash of an object to a time
                         -- Key usage bits that may be consistent: digitalSignature
                         -- and/or nonRepudiation

   id-kp-OCSPSigning     OBJECT IDENTIFIER ::= { id-kp 9 }
                         -- Signing OCSP responses
                         -- Key usage bits that may be consistent: digitalSignature
                         -- and/or nonRepudiation
```

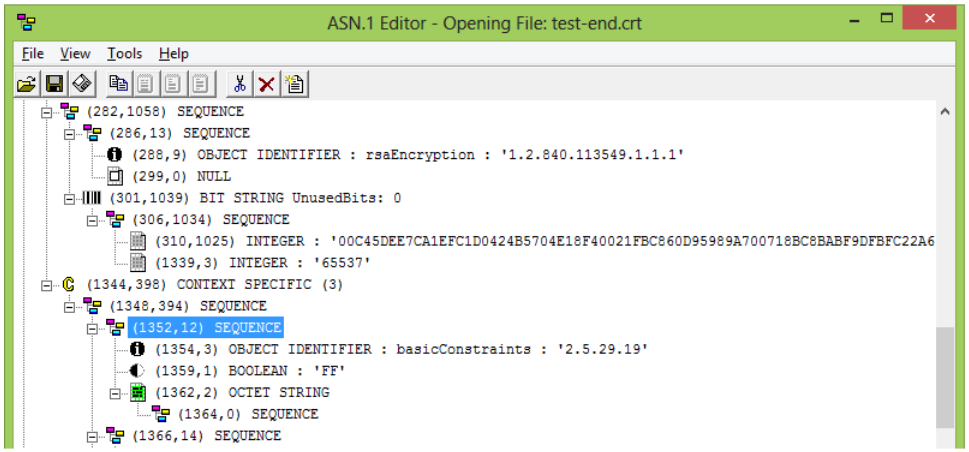


X.509 anatomy

basicConstraints




```
basicConstraints

   id-ce-basicConstraints OBJECT IDENTIFIER ::= { id-ce 19 }

   BasicConstraints ::= SEQUENCE {
                                     cA                 BOOLEAN DEFAULT FALSE,
                                     pathLenConstraint  INTEGER (0..MAX) OPTIONAL
                                   }
```



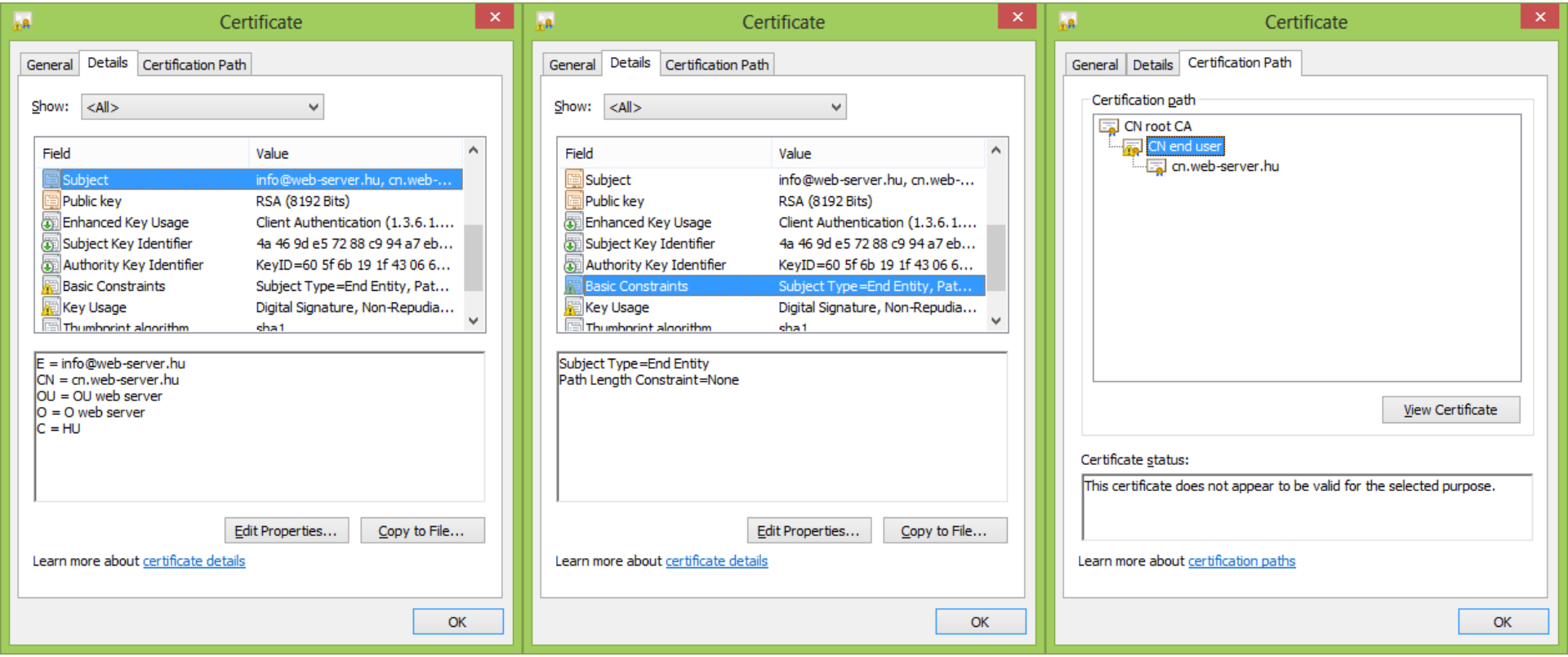
X.509 anatomy

basicConstraints and Moxie Marlinspike

```
openssl req
                -config ./cryptonite/openssl.conf
                -new
                -out ./cryptonite/test-web.csr
                -newkey rsa:8192

openssl rsa
                -in privkey.pem
                -out ./cryptonite/test-web.key

openssl x509
                -in ./cryptonite/test-web.csr
                -out ./cryptonite/test-web.crt
                -req
                -days 1826
                -CA ./cryptonite/test-end.crt
                -CAkey ./cryptonite/test-end.key
                -CAcreateserial
                -CAserial ./cryptonite/test-web.seq
                -extfile ./cryptonite/test-web.ext
                -sha256

openssl pkcs12
                -export
                -in ./cryptonite/test-web.crt
                -inkey ./cryptonite/test-web.key
                -CSP "Microsoft Enhanced RSA and AES Cryptographic Provider"
                -certfile ./cryptonite/test-root.crt
                -certfile ./cryptonite/test-end.crt
                -out ./cryptonite/test-web.p12
```

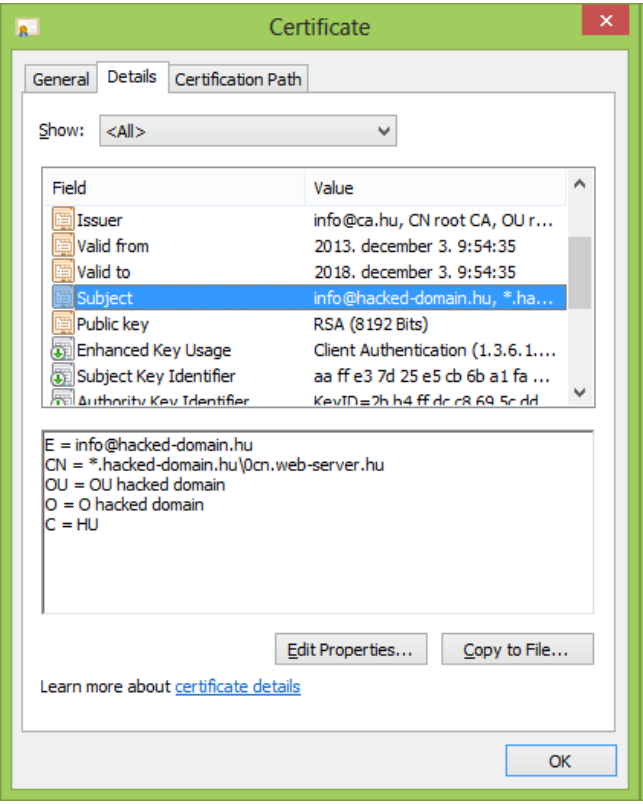
X.509 anatomy

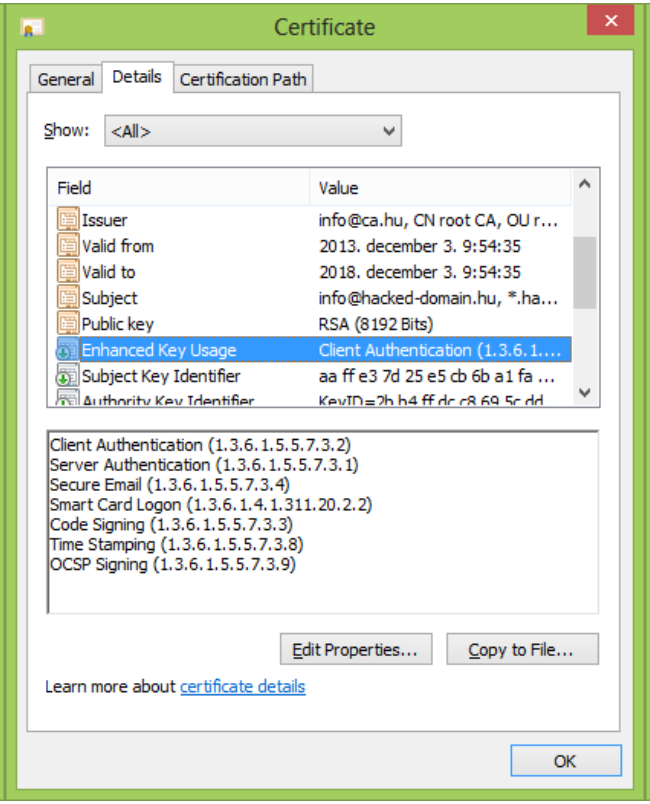
basicConstraints and Moxie Marlinspike

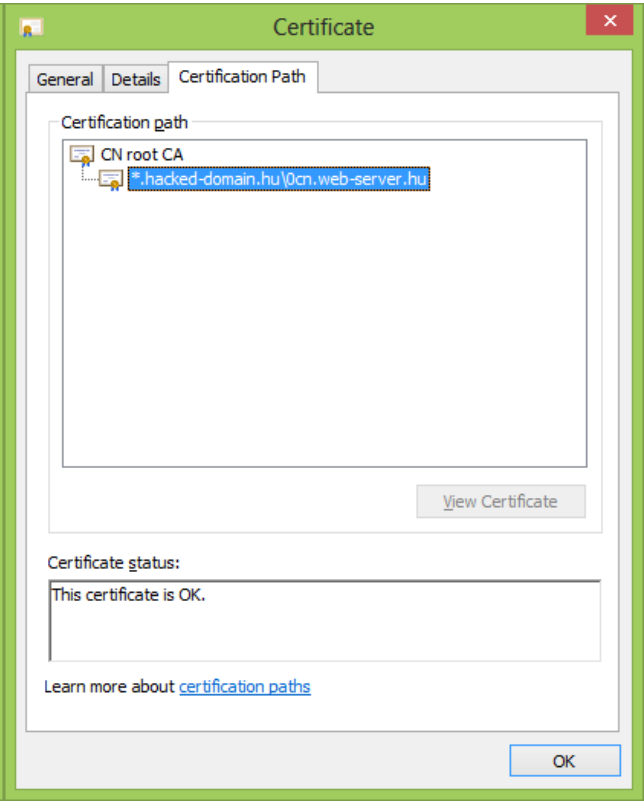


X.509 anatomy

subject „\0” and Moxie Marlinspike

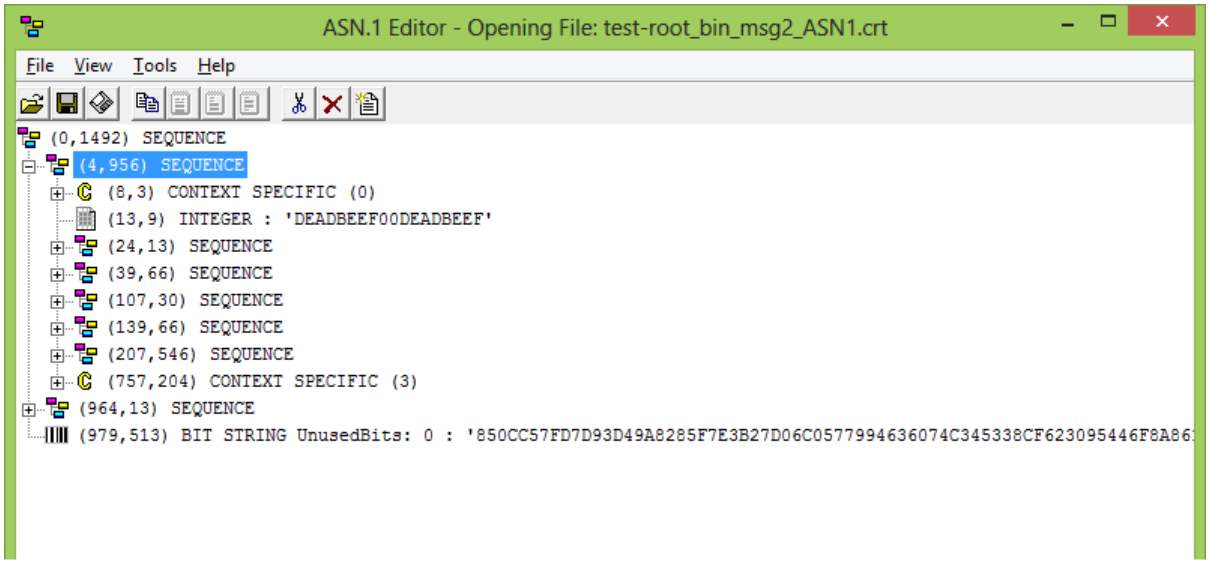






X.509 anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



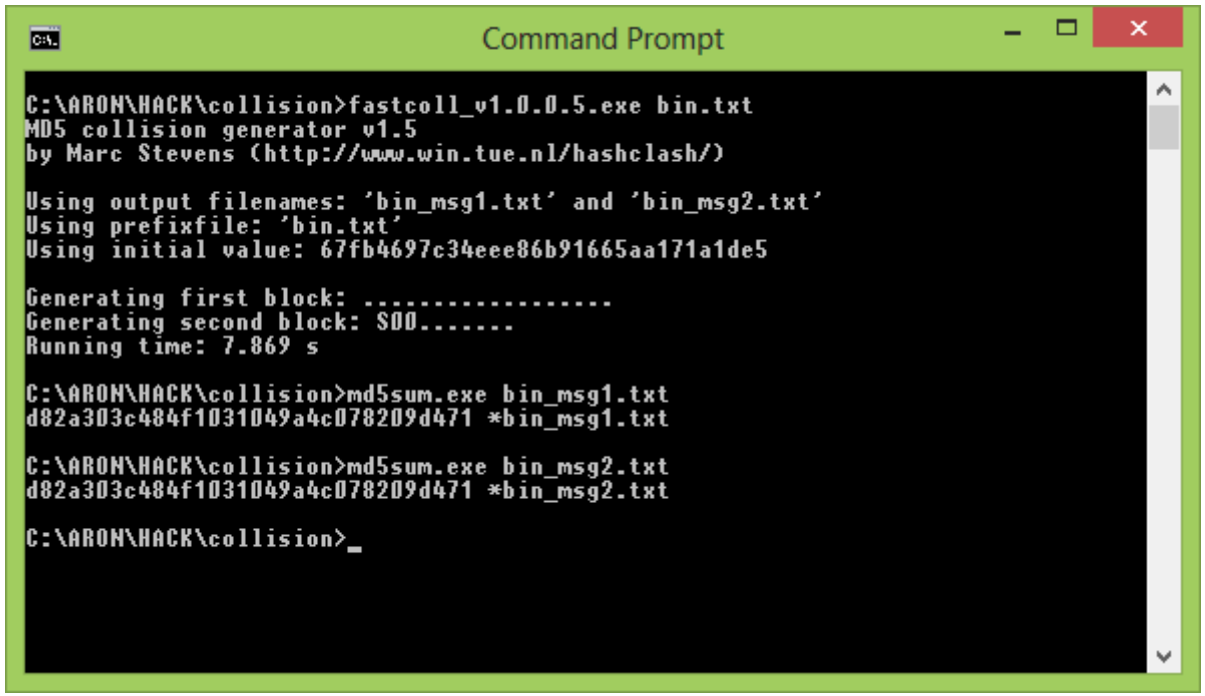
Certificate and TBSCertificate

```
Certificate         ::= SEQUENCE {
   tbsCertificate                    TBSCertificate,
   signatureAlgorithm                AlgorithmIdentifier,
   signatureValue                    BIT STRING
                                   }

TBSCertificate      ::= SEQUENCE {
   version                             [0] EXPLICIT Version DEFAULT v1,
   serialNumber                        CertificateSerialNumber,
   signature                           AlgorithmIdentifier,
   issuer                              Name,
   validity                            Validity,
   subject                             Name,
   subjectPublicKeyInfo                SubjectPublicKeyInfo,
   issuerUniqueID                    [1] IMPLICIT UniqueIdentifier OPTIONAL,
   subjectUniqueID                   [2] IMPLICIT UniqueIdentifier OPTIONAL,
   extensions                        [3] EXPLICIT Extensions OPTIONAL
                                   }

AlgorithmIdentifier ::= SEQUENCE {
   algorithm                           OBJECT IDENTIFIER,
   parameters                          ANY DEFINED BY algorithm OPTIONAL
                                   }
```

X.509 anatomy

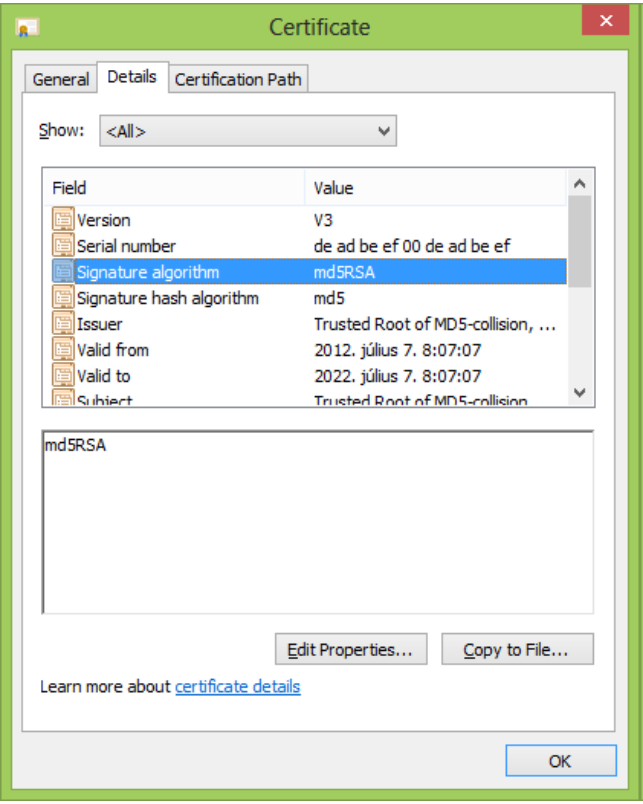
MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima

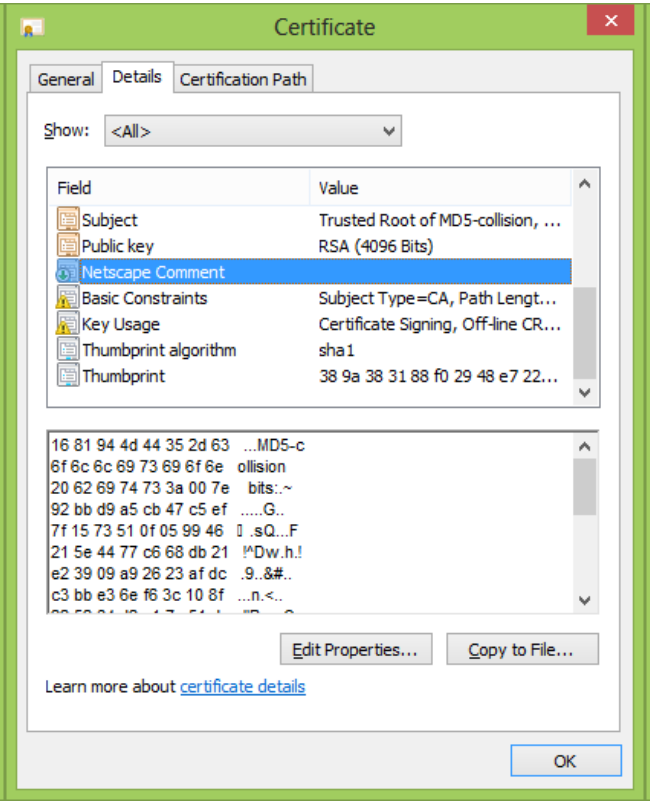
- add collision bits to tbsCertificate (bin.txt → bin_msg1.txt + bin_msg2.txt)
- add/copy SEQUENCE, signatureAlgorithm, signatureValue to tbsCertificate

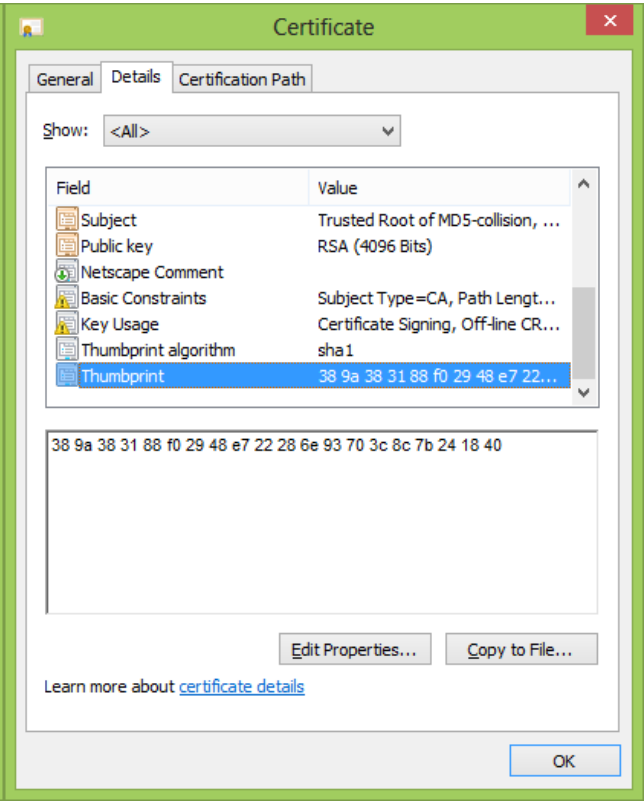


X.509 anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima

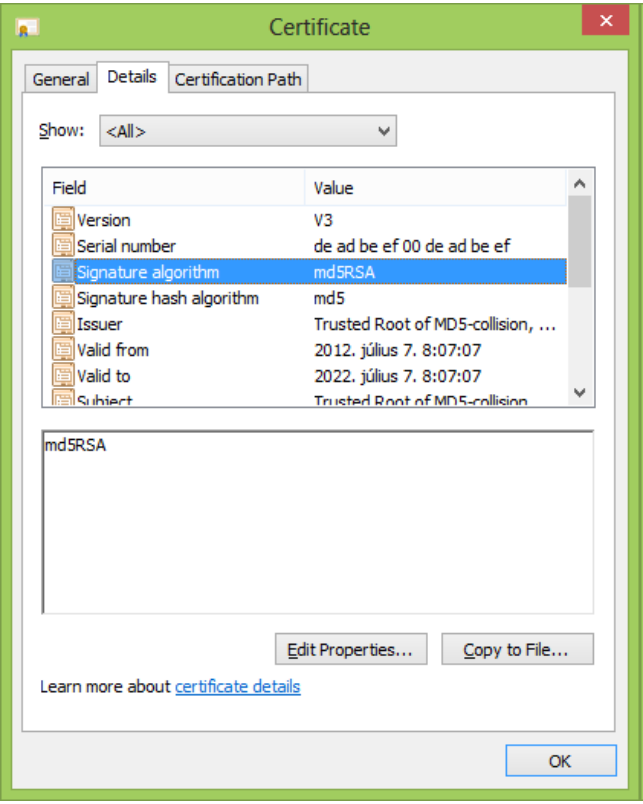


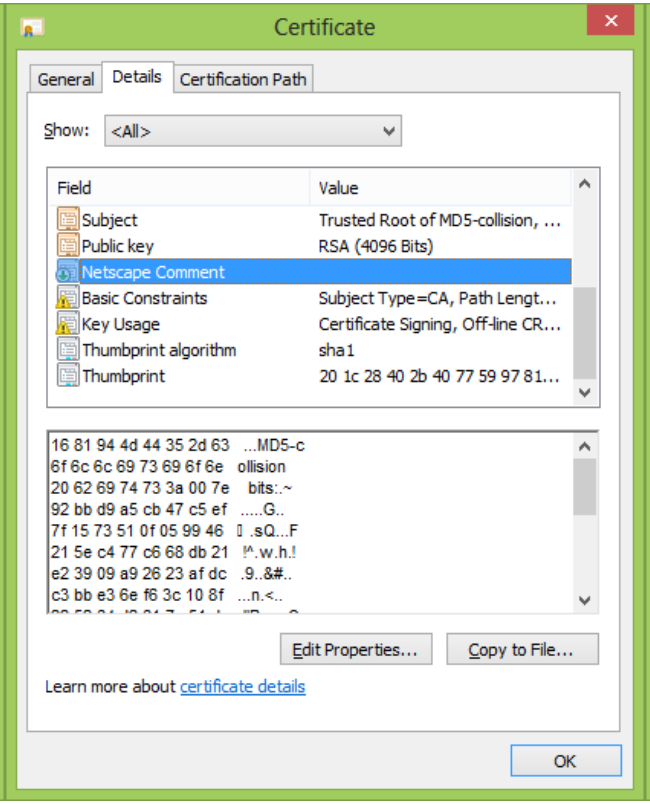


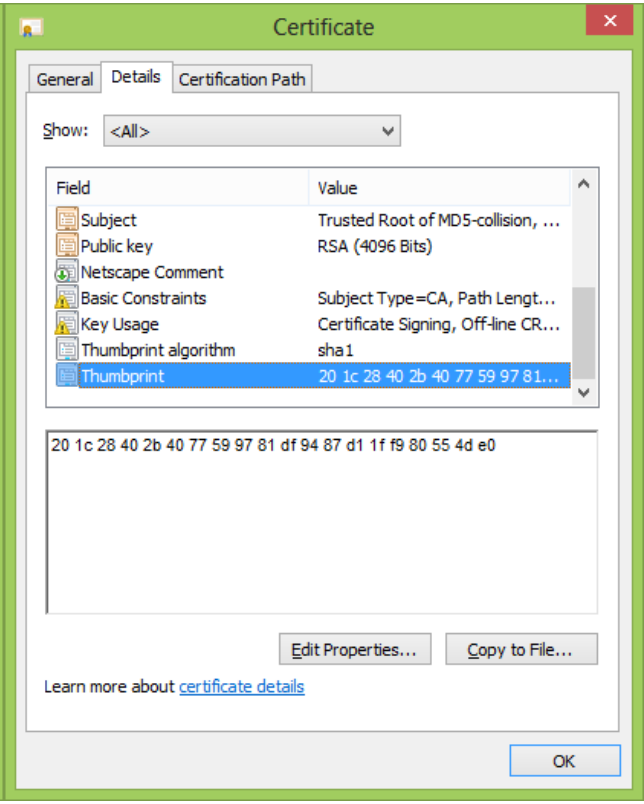


X.509 anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima

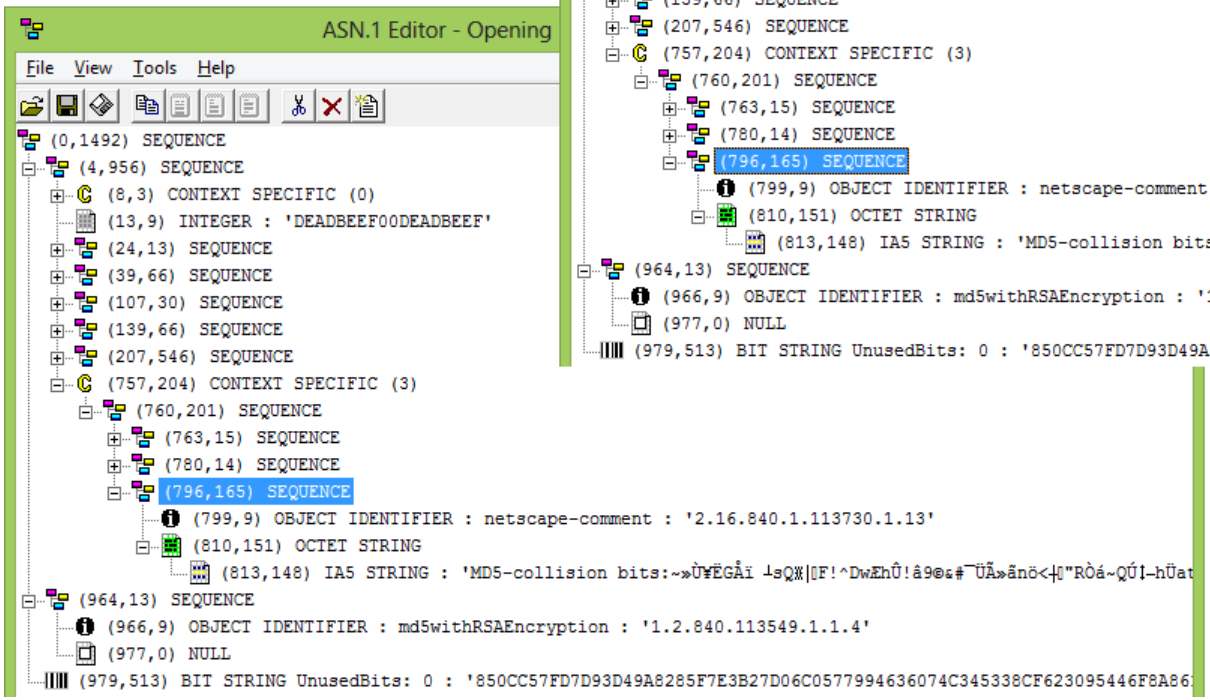






X.509 anatomy

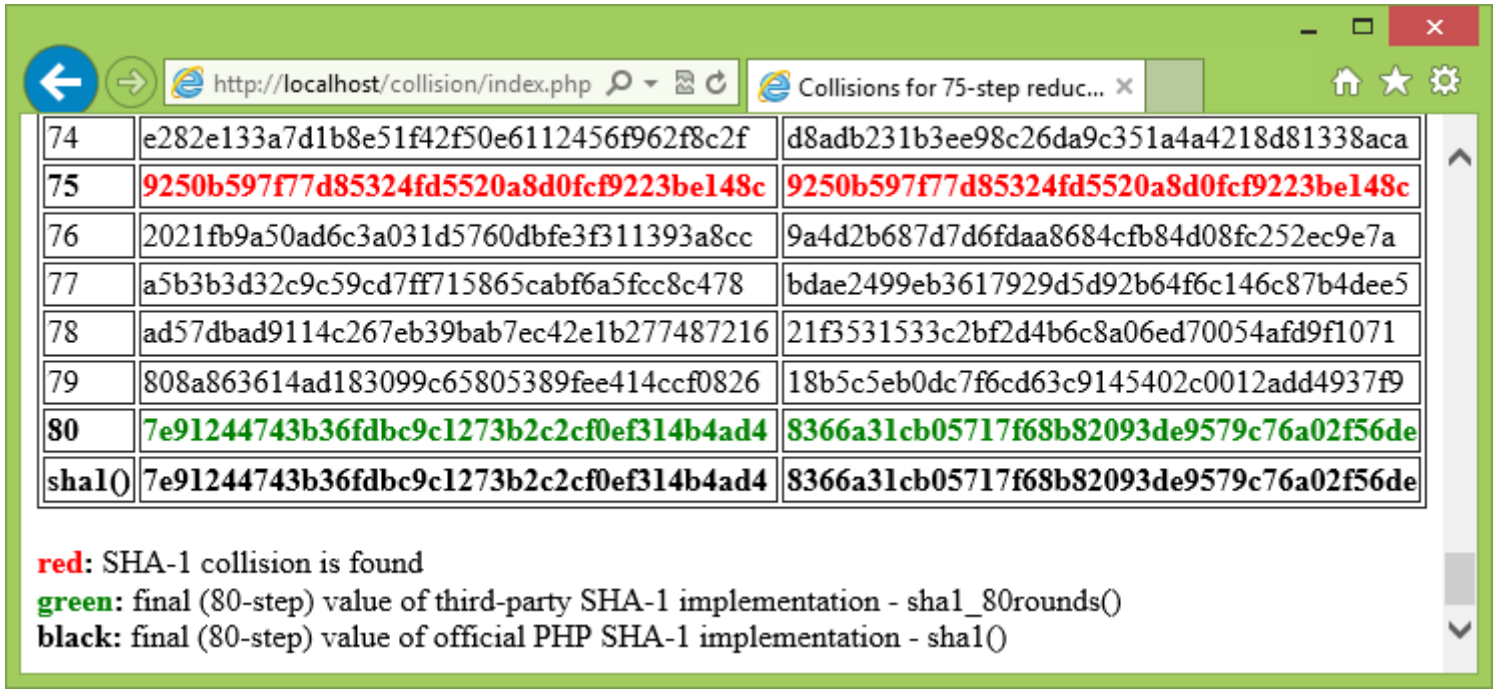
MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



X.509 anatomy

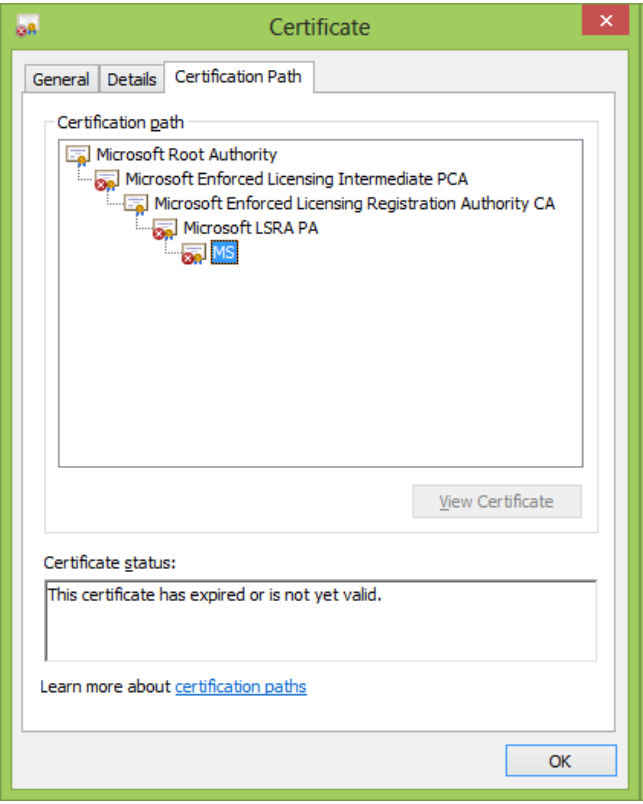
MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima

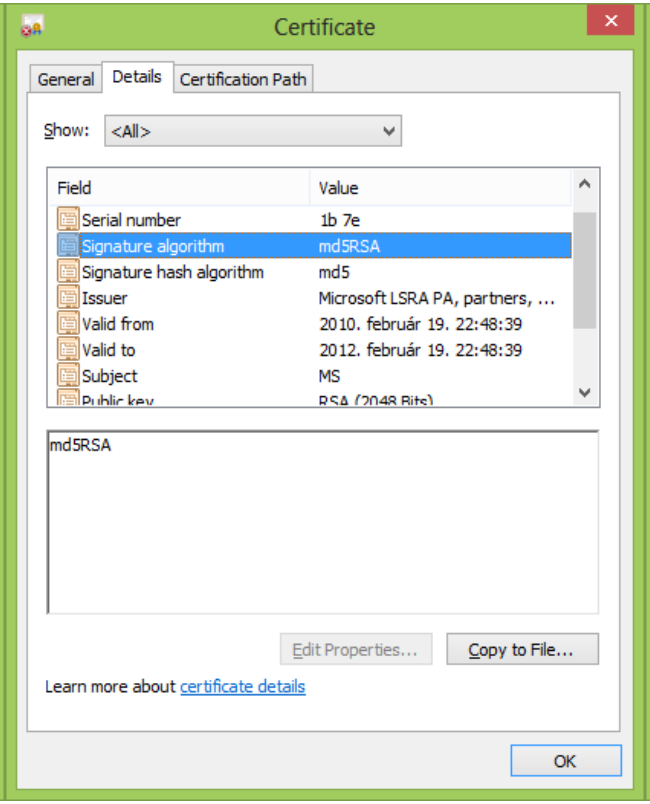
- Meanwhile at the team of E. A. Grechnikov: collisions for 75-step reduced SHA-1
- NIST selected Keccak (2012-10-02) as the basis for SHA-3

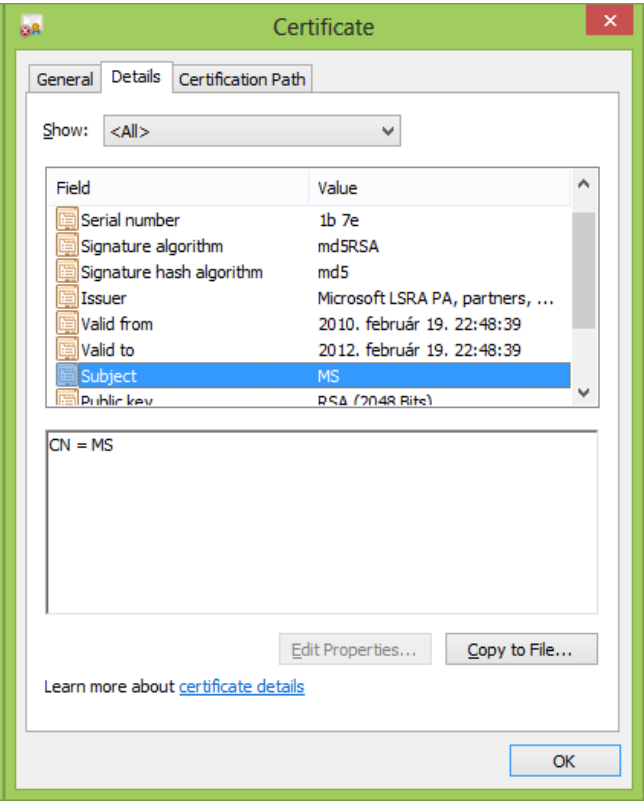


X.509 anatomy

MD5 collision and Flame, WuSetupV.exe



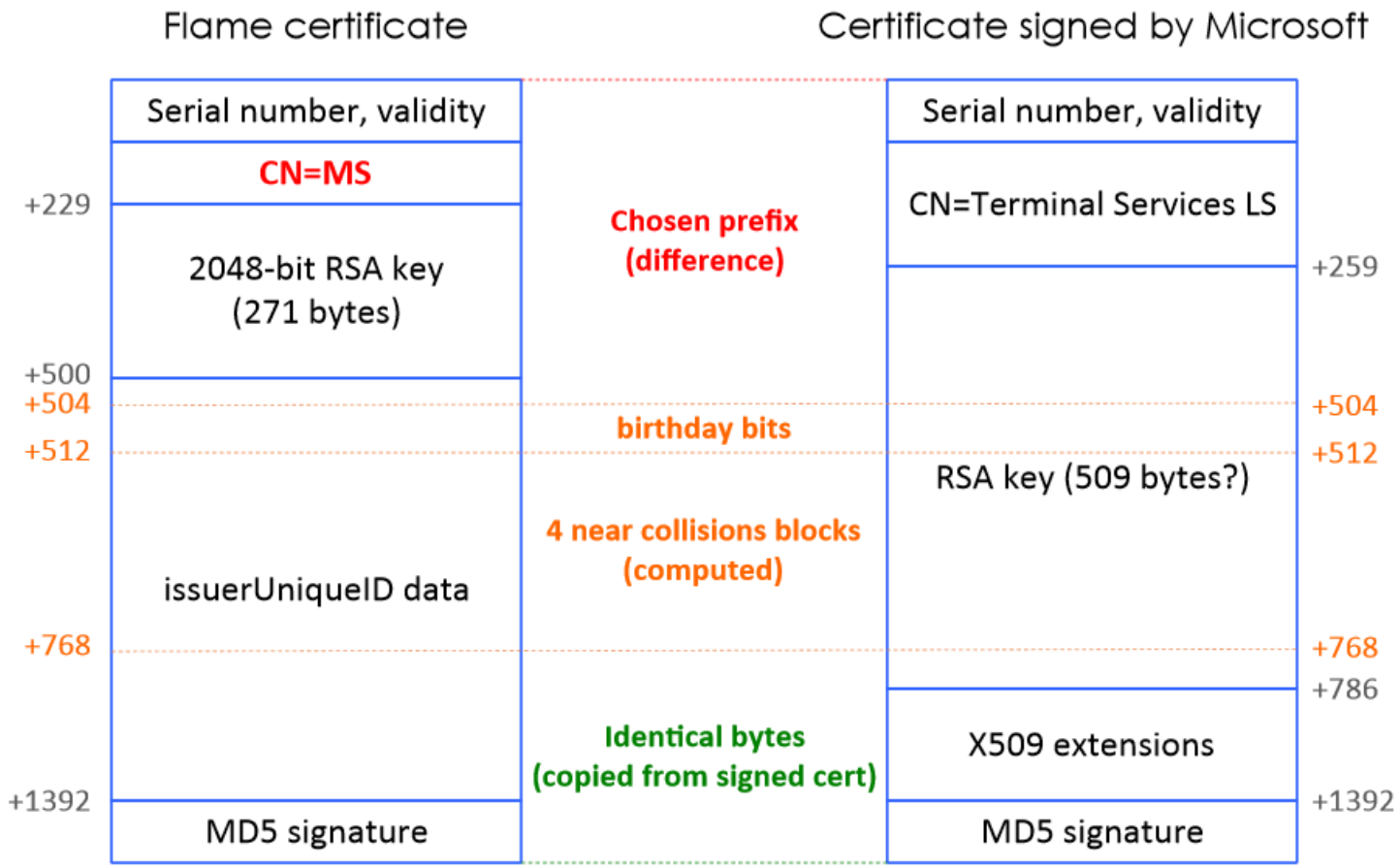




X.509 anatomy

MD5 collision and Flame, WuSetupV.exe

- Marc Stevens: chosen-prefix collisions
- Alex Sotirov: analysis of Flame certificate



Windows and cryptography

We are working with smart cards, cryptography tools since 1993. Why are these hot topics in Hungary nowadays?

- new (simplified) regulation for digital signatures in Hungarian e-government

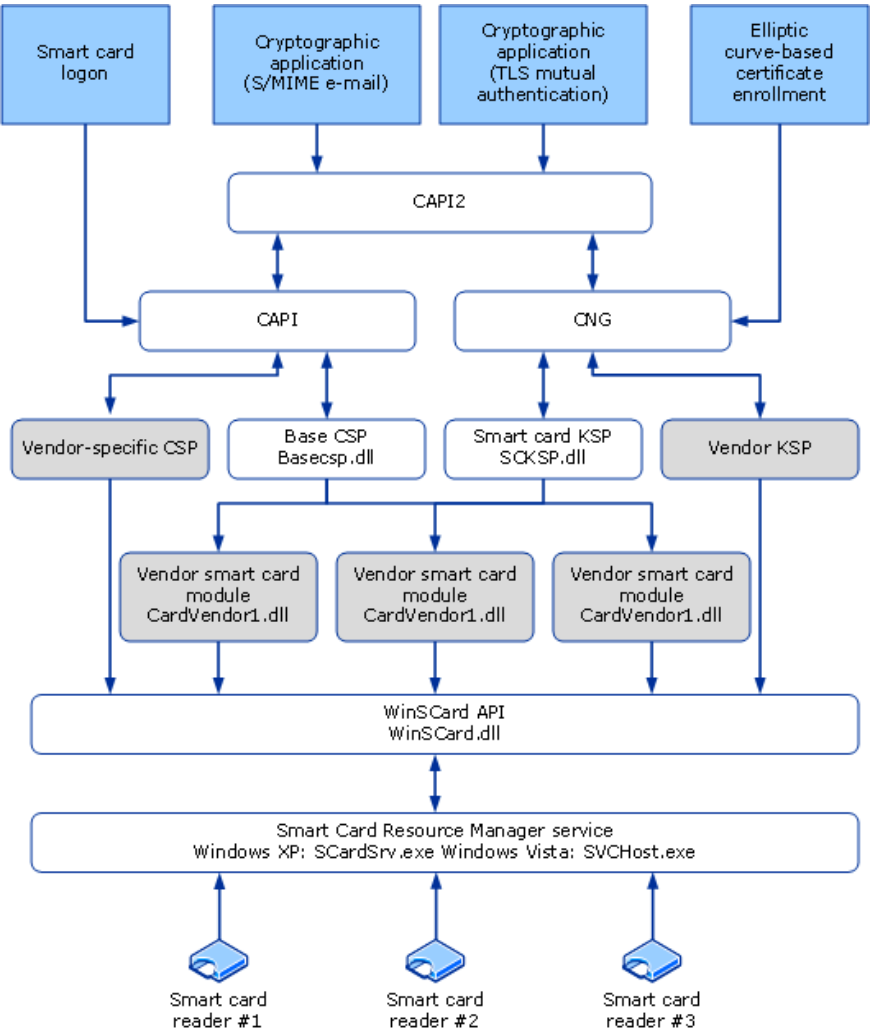
  using software tokens with qualified certificates (PKCS#12: .p12/.pfx files) for creating advanced electronic signatures

  (see: Governmental Decree No. 78/2010. (III. 25.) Section 5. (2))

- new contactless cards for students, mass transit and e-government

  some of them contains PKI (e.g.: SmartMX)

Windows and cryptography



source: microsoft.com
(http://msdn.microsoft.com/en-us/library/bb905527.aspx)

API (Application Programming Interface):
high-level interface for developers
e.g. CryptSignHash(), CryptSignMessage()
(MS CryptoAPI, CNG, PKCS#11)

CSP (Cryptographic Service Provider),
KSP (Key Storage Provider):
HW and SW token driver from vendor
e.g. addressing private keys and key slots
(Gemalto, Oberthur, G&D)

APDU (Application Programming Data Unit):
general rules for data structures
e.g. 00 B0 00 00 FF
(ISO/IEC 7816-4 APDUs or pseudo-APDUs)

HW token reader:
WinSCard.dll: selects remote or local service
SCardSrv.exe: selects device and its interface
e.g. native interface of registered devices
(PC/SC interface is common)

Windows certificate stores

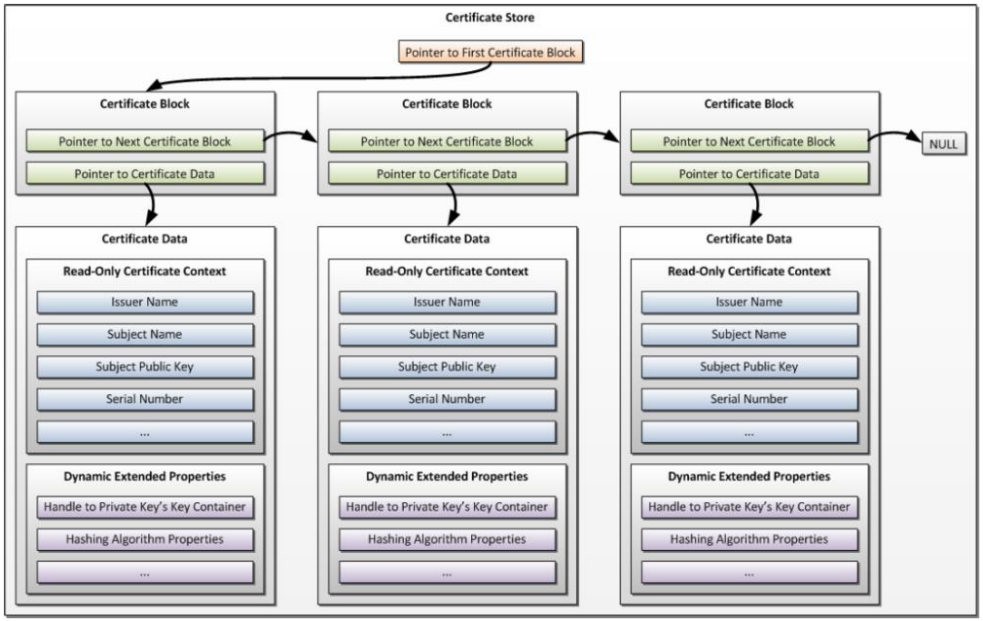
„Let's dump the protected imported private keys from a Windows Server!"

Windows certificate stores

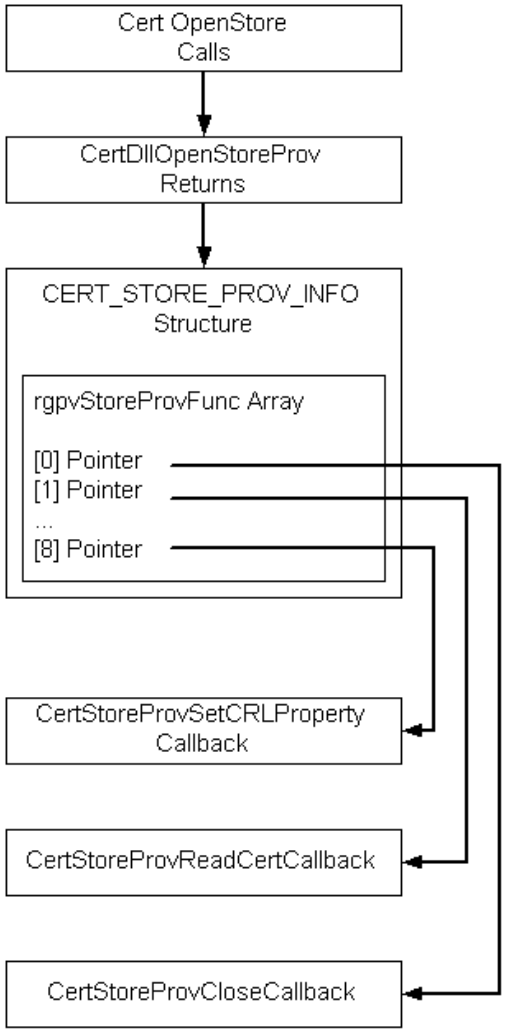
What does Microsoft tell us about CertOpenStore?

„The store provider function copies its certificates [...] to the in-memory store [...]. The new store provider function can use any of the CryptoAPI [...] functions, [...], to add its Certificates and CRLs to the in-memory store."

... but also copies CRYPT_EXPORTABLE flag of private keys!!!
... and these flags can be modified!!!



source: blackhat.com
(https://media.blackhat.com/bh-eu-11/Jason_Geffner/BlackHat_EU_2011_Geffner_Exporting_RSA_Keys-WP.pdf)



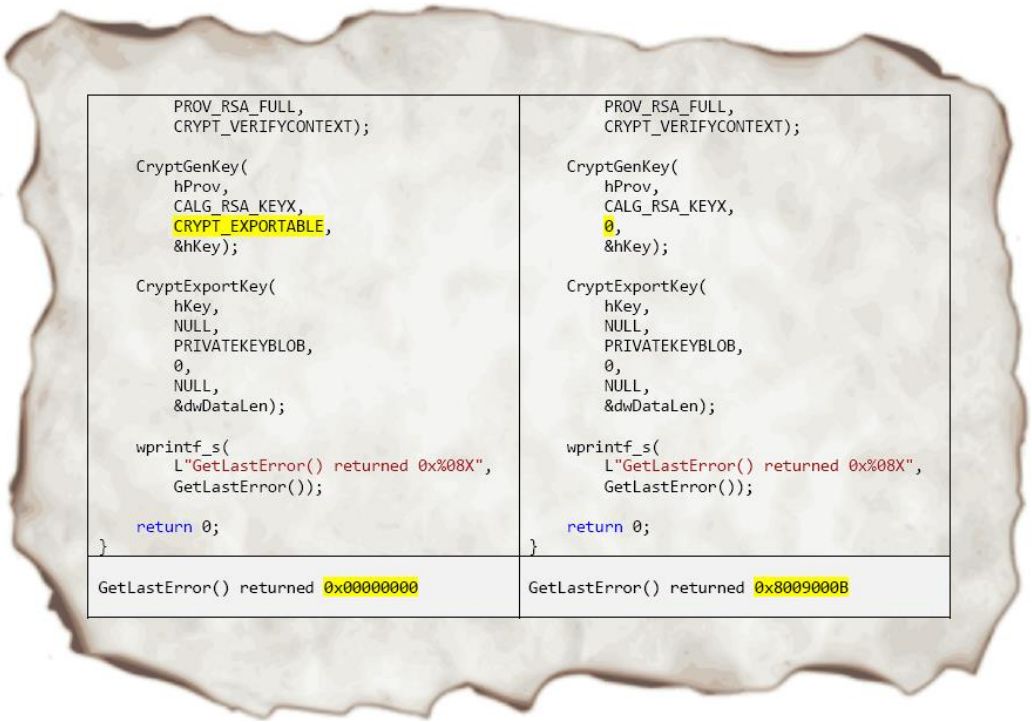
source: microsoft.com
(http://msdn.microsoft.com/en-us/library/windows/desktop/aa382403.aspx)

Windows certificate stores

Jason Geffner (March 18, 2011) at Black Hat Europe 2011 also talked about this issue, but his Proof-of-Concept code covered just newer operating systems:

- ncrypt.dll is needed

  Windows Vista/7
  Windows Server 2008/2008 R2
  2012/2012 R2



```
        PROV_RSA_FULL,
        CRYPT_VERIFYCONTEXT);

    CryptGenKey(
        hProv,
        CALG_RSA_KEYX,
        CRYPT_EXPORTABLE,
        &hKey);

    CryptExportKey(
        hKey,
        NULL,
        PRIVATEKEYBLOB,
        0,
        NULL,
        &dwDataLen);

    wprintf_s(
        L"GetLastError() returned 0x%08X",
        GetLastError());

    return 0;
}
```

GetLastError() returned 0x00000000

```
        PROV_RSA_FULL,
        CRYPT_VERIFYCONTEXT);

    CryptGenKey(
        hProv,
        CALG_RSA_KEYX,
        0,
        &hKey);

    CryptExportKey(
        hKey,
        NULL,
        PRIVATEKEYBLOB,
        0,
        NULL,
        &dwDataLen);

    wprintf_s(
        L"GetLastError() returned 0x%08X",
        GetLastError());

    return 0;
}
```

GetLastError() returned 0x8009000B

source: blackhat.com
(https://media.blackhat.com/bh-eu-11/Jason_Geffner/BlackHat_EU_2011_Geffner_Exporting_RSA_Keys-WP.pdf)

Our solution also works on other (older) operating systems as well!

- crypt32.dll is needed

  Windows XP/Vista/7/8
  Windows Server 2003/2003 R2/2008/2008 R2/2012/2012 R2

Windows certificate stores

How can our tool export protected keys?

- CertOpenStore(), CertEnumCertificatesInStore()

  creates a copy of certificate store (without private keys)
  makes a list of stored certificates and their properties

- CryptAcquireContext(), CryptAcquireCertificatePrivateKey()

  sets CSP and CRYPT_SILENT or CRYPT_ACQUIRE_SILENT_FLAG flags

- CryptGetUserKey()

  gets handle that manages private key (in a CSP) for each listed certificate

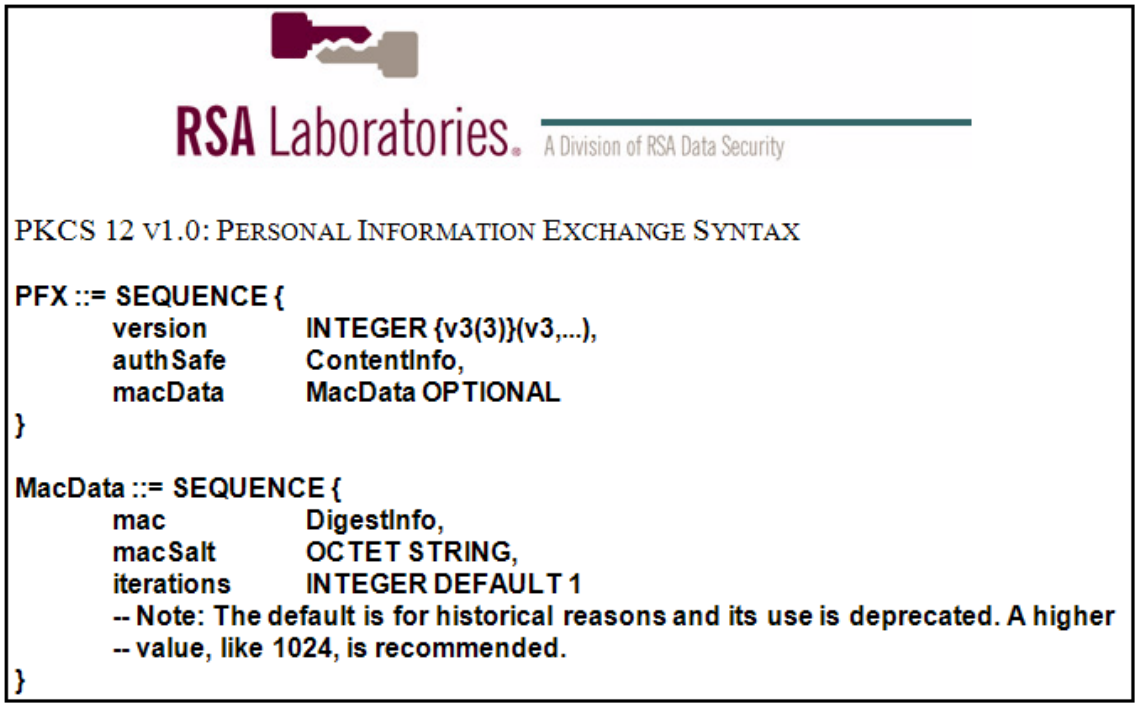
- CRYPT_EXPORTABLE, CryptExportKey()

  sets CRYPT_EXPORTABLE flag in copy of certificate store (memory)
  gets PRIVATEKEYBLOB from a separate store (PKCS#12 – .pfx/.p12 files)

Windows certificate stores

What can be the countermeasures?

- use HW tokens for storing private keys (if it is possible)
- do not copy CRYPT_EXPORTABLE flags (Microsoft should fix it)
- „Enable strong private key protection" of SW tokens is not enough (SILENT)
  use PKCS#12 (.p12/.pfx) SW tokens that really store encrypted private keys



Windows certificate stores

PIN/password-management

„Let's play with PIN/passwords of HW tokens!"

PIN/password-management

What does Microsoft tell us about PIN/password cache?

„The Base CSP internally maintains a per-process cache of the PIN to enable caching. The PIN is stored encrypted in memory."

source: microsoft.com
(http://msdn.microsoft.com/en-us/library/bb905527.aspx)

CWA 14169

March 2004

Secure signature-creation devices "EAL 4+"

5.1.3.3 Timing of authentication (FIA_UAU.1)

FIA_UAU.1.1 The TSF shall allow [
1. Identification of the user by means of TSF required by FIA_UID.1.
2. Establishing a user trusted channel between the TOE and a SSCD of Type 3 by means of TSF required by FTP_ITC.1/SSCD import.
3. Establishing a trusted path between local user and the TOE by means of TSF required by FTP_TRP.1/TOE.
4. Establishing a trusted channel between the SCA and the TOE by means of TSF required by FTP_ITC.1/DTBS import.]
on behalf of the user to be performed before the user is authenticated.

FIA_UAU.1.2 The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

CERTIFIED

... which means, that in one session the PIN/password can be cached.

... but Common Criteria EAL 4+ evaluated HW tokens have to enforce successful authentication (and user interaction) before each function call!

PIN/password cache capability depends on CSP!

source: cenorm.be
(ftp://ftp.cenorm.be/Public/Cwas/e-europe/esign/cwa14169-00-2004-Mar.pdf)

PIN/password-management

We assume that CSPs does not keep PIN/password in cache. Can we still automate PIN/password setting for each signature creation? What function shall we use?

- CryptSetProvParam()

  The application can cache the PIN/password and use it in the background!

```
BOOL WINAPI CryptSetProvParam(
  __in  HCRYPTPROV hProv,
  __in  DWORD dwParam,
  __in  const BYTE *pbData,
  __in  DWORD dwFlags
);
```

PP_KEYEXCHANGE_PIN 32 (0x20)	Specifies that the key exchange PIN is contained in *pbData*. The PIN is represented as a null-terminated ASCII string.
PP_PIN_PROMPT_STRING 44 (0x2C)	Sets an alternate prompt string to display to the user when the user's PIN is requested. The *pbData* parameter is a pointer to a null-terminated Unicode string that contains the string.
PP_SIGNATURE_PIN 33 (0x21)	Specifies that the signature PIN is contained in *pbData*. The PIN is represented as a null-terminated ASCII string.
PP_SECURE_KEYEXCHANGE_PIN 47 (0x2F)	Specifies that an encrypted key exchange PIN is contained in *pbData*. The *pbData* parameter contains a **DATA_BLOB**.
PP_SECURE_SIGNATURE_PIN 48 (0x30)	Specifies that an encrypted signature PIN is contained in *pbData*. The *pbData* parameter contains a **DATA_BLOB**.

source: microsoft.com
(http://msdn.microsoft.com/en-us/library/windows/desktop/aa380276.aspx)

PIN/password-management

How can our tool use PIN/password of HW tokens automatically in the background?

- CryptAcquireContext()

  retrieves handle of CSP that contains private key of chosen certificate

- CryptSetProvParam()

  sets given PIN/password – since NTDDI_WINXPSP2 – (using handle of CSP) for PP_SECURE_KEYEXCHANGE_PIN or PP_SECURE_SIGNATURE_PIN

- CryptCreateHash()

  initializes and returns handle of hash value

- CryptHashData()

  sets data to be hashed (using handle of hash value)

- CryptSignHash()

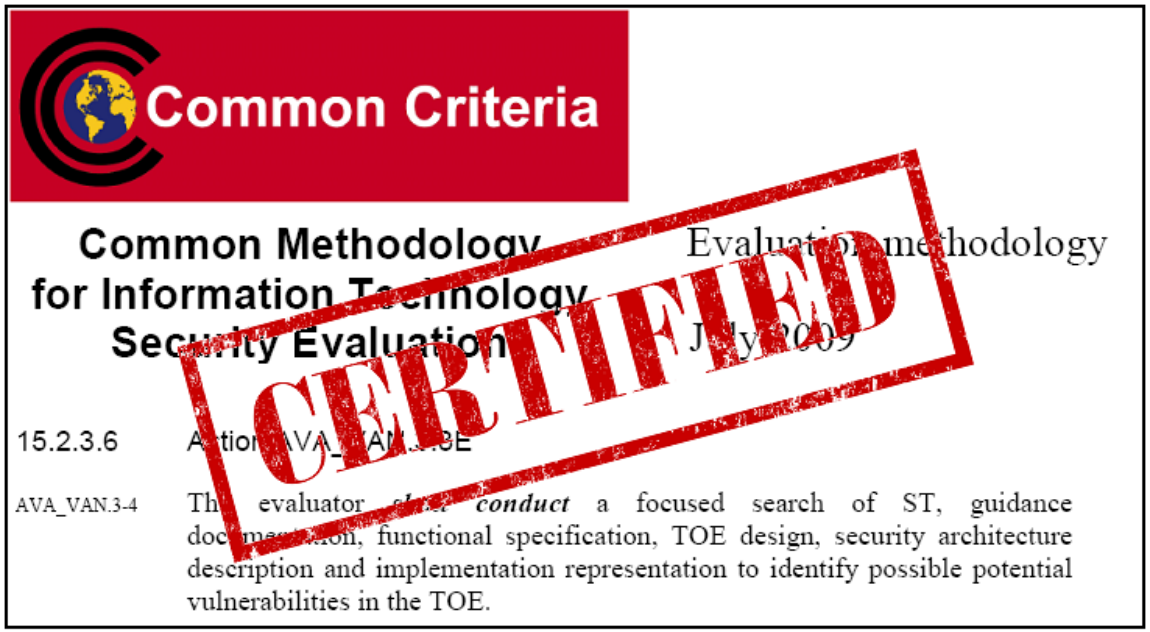
  creates signature

PIN/password-management

What can be the countermeasures?

- we can not clearly decide whether PIN/password cache is „good” or „bad” (e.g. imagine that someone has to sign digitally hundreds of documents a day – after visual verification of contents - using a smart card with PIN/password)

  ... but if you need this functionality, be sure that either signature-creation application or CSP manages PIN/password values in a secure way



(e.g. source code analysis at Common Criteria EAL4 level or above)

source: commoncriteriaportal.org (http://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R3.pdf)

Communication channels

„Let's check the digital signatures of the CSP layer!"

Communication channels

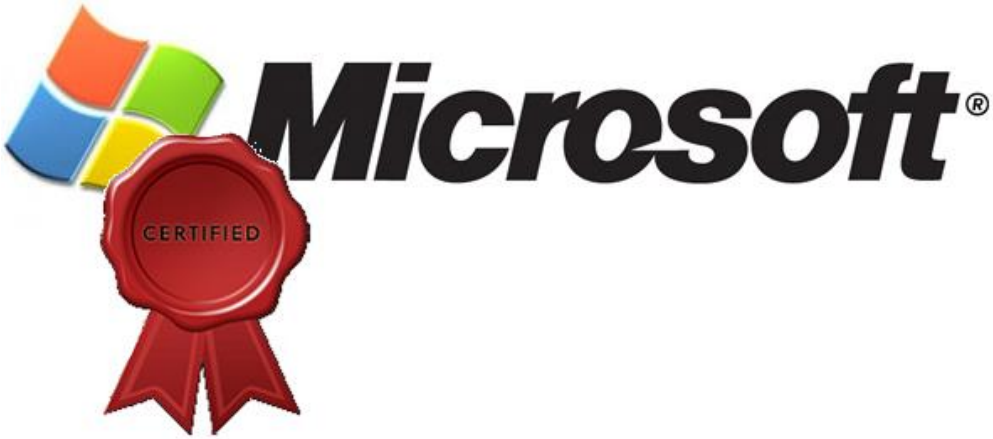
What do we know about secure HW tokens?

- most of them are certified based on FIPS or Common Criteria
  e.g. G&D SmartCafe Expert 3.2 FIPS 140-2 level 3, Common Criteria EAL 5+
  e.g. such certification costs about $250.000

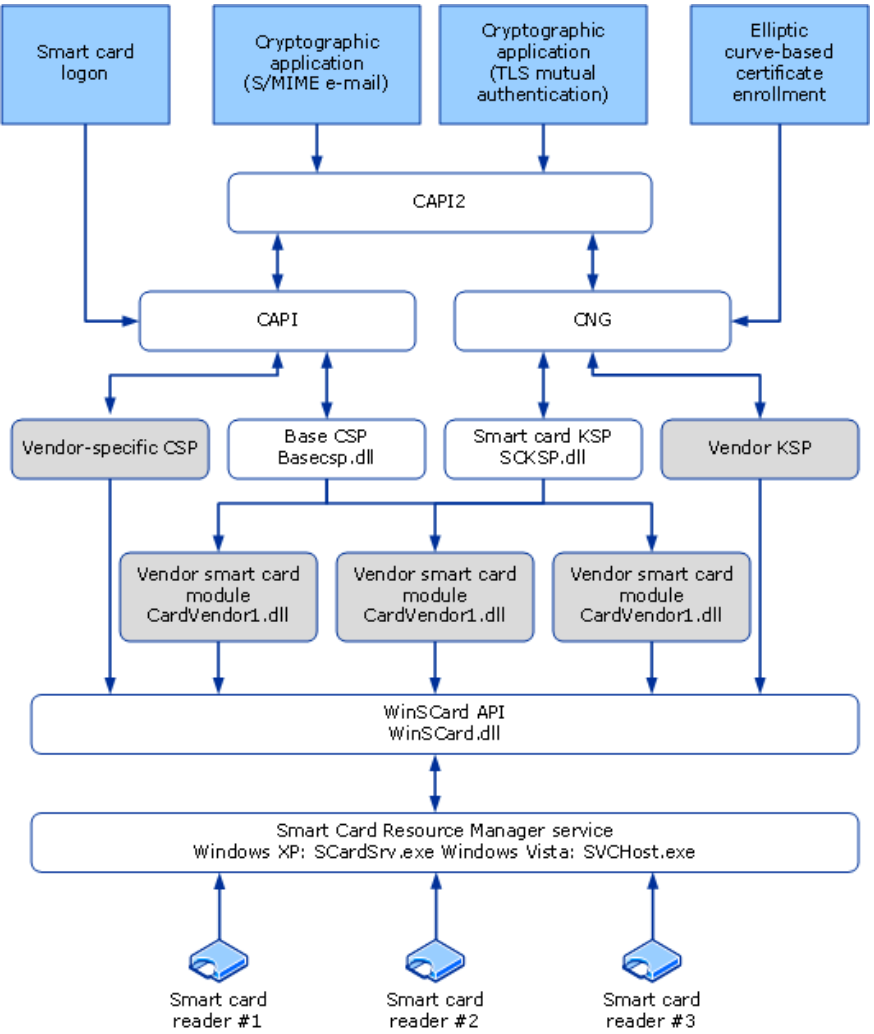
  ... but these certifications cover just the HW tokens themselves!
  ... in most cases they do not tell us anything about running environment!

The CSPs are protected: they are signed by Microsoft!

... but is it enough?



Communication channels



source: microsoft.com
(http://msdn.microsoft.com/en-us/library/bb905527.aspx)

The CSPs are protected: they are signed by Microsoft!

„*Vendors can develop hardware or software CSPs that support a wide range of cryptographic operations and technologies. However, Microsoft must certify and digitally sign all CSPs.*"

source: microsoft.com
(http://technet.microsoft.com/en-us/library/cc776447.aspx)

CSP signature is an extra security layer on .dll files which is created by cspSign.exe (separately stored .sig file or embedded into resource file).

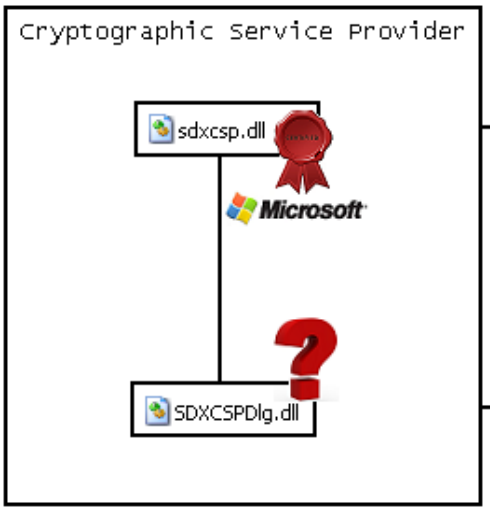
Communication channels

How can we change parts of a signed CSP in the background?

- all .dll files of CSP are signed by Microsoft, but
  just registered .dll file of CSP is verified by Windows!

  dependency tree of .dll files of CSP is not checked

In our real-life experience we sent all parts (sdxcsp.dll and SDXCSPDlg.dll) of our CSP to Microsoft to be signed. The sdxcsp.dll file was set in the registry, but this file imported also the SDXCSPDlg.dll (in which content could be modified)!




```
#import "..\SDXCSPDlg.dll"
using namespace SDXCSPDLGLib;
```

```
BOOL WINAPI CSDXCSPClass::CPDestroyKey(HCRYPTPROV hProv, HCRYPTKEY hKey)
{
        return m_comPtr->CPDestroyKey(hProv, hKey);
}
```

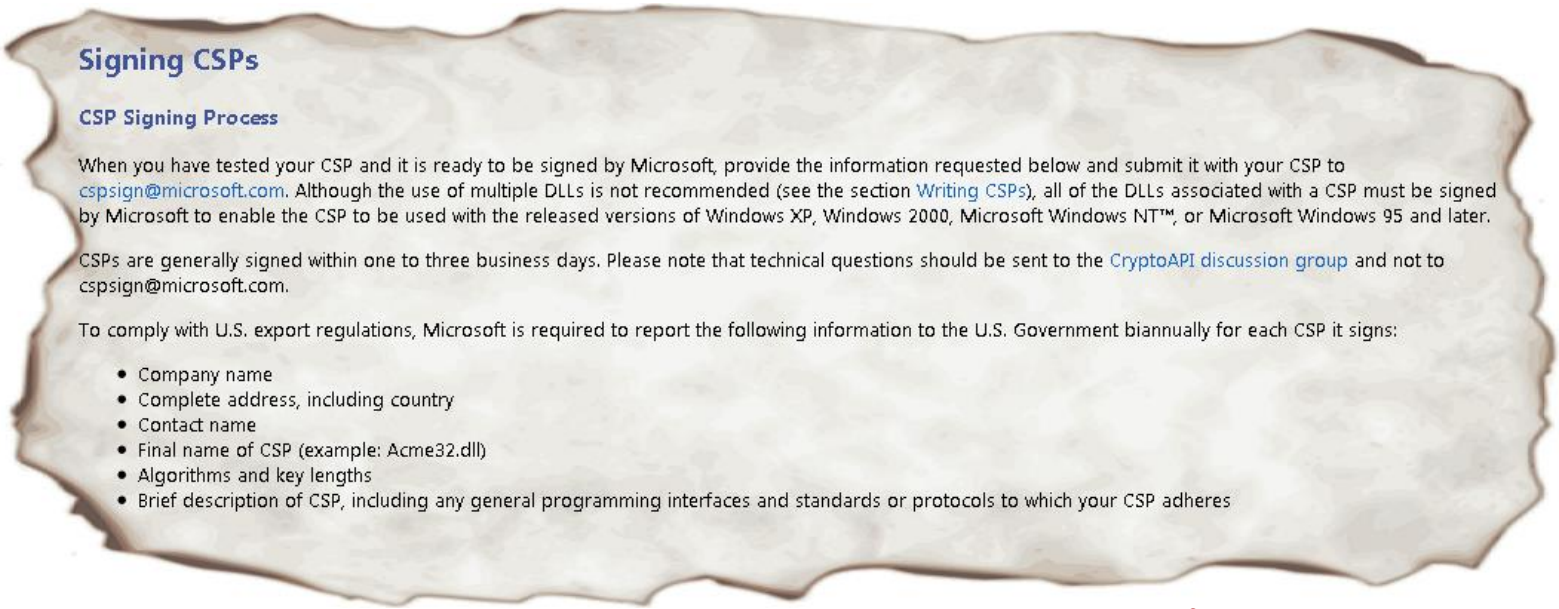
Communication channels

What can be the countermeasures?

- explicitly verify all .dll files of CSP loaded into memory

  e.g. tools can be used that verify hashes of files based on „white lists”

- enforce verification of all CSP files by Windows (Microsoft should fix it)



Signing CSPs

CSP Signing Process

When you have tested your CSP and it is ready to be signed by Microsoft, provide the information requested below and submit it with your CSP to cspsign@microsoft.com. Although the use of multiple DLLs is not recommended (see the section Writing CSPs), all of the DLLs associated with a CSP must be signed by Microsoft to enable the CSP to be used with the released versions of Windows XP, Windows 2000, Microsoft Windows NT™, or Microsoft Windows 95 and later.

CSPs are generally signed within one to three business days. Please note that technical questions should be sent to the CryptoAPI discussion group and not to cspsign@microsoft.com.

To comply with U.S. export regulations, Microsoft is required to report the following information to the U.S. Government biannually for each CSP it signs:

- Company name
- Complete address, including country
- Contact name
- Final name of CSP (example: Acme32.dll)
- Algorithms and key lengths
- Brief description of CSP, including any general programming interfaces and standards or protocols to which your CSP adheres

source: microsoft.com
(http://msdn.microsoft.com/en-us/library/ms953432.aspx)

... but at CNG (Cryptography API: Next Generation) where KSPs (Key Storage Provider) can be created, it seems, that these rules will change!

Communication channels

„Let's dump the communication of HW tokens!"

Communication channels

The communication channels between HW tokens and their CSPs should be protected...

„PC/SC functionality is exposed to applications via the Windows Smart Card (WinSCard) client API, implemented in winscard.dll and, to a lesser degree, scarddlg.dll. [...] Each command is sent to the card via the WinSCard function SCardTransmit."

source: microsoft.com
(http://msdn.microsoft.com/en-us/magazine/cc163521.aspx)

... but winscard.dll can be replaced without any error!
... communication between CSP and smart card can be monitored!
... and in most cases these communication channels are not encrypted!

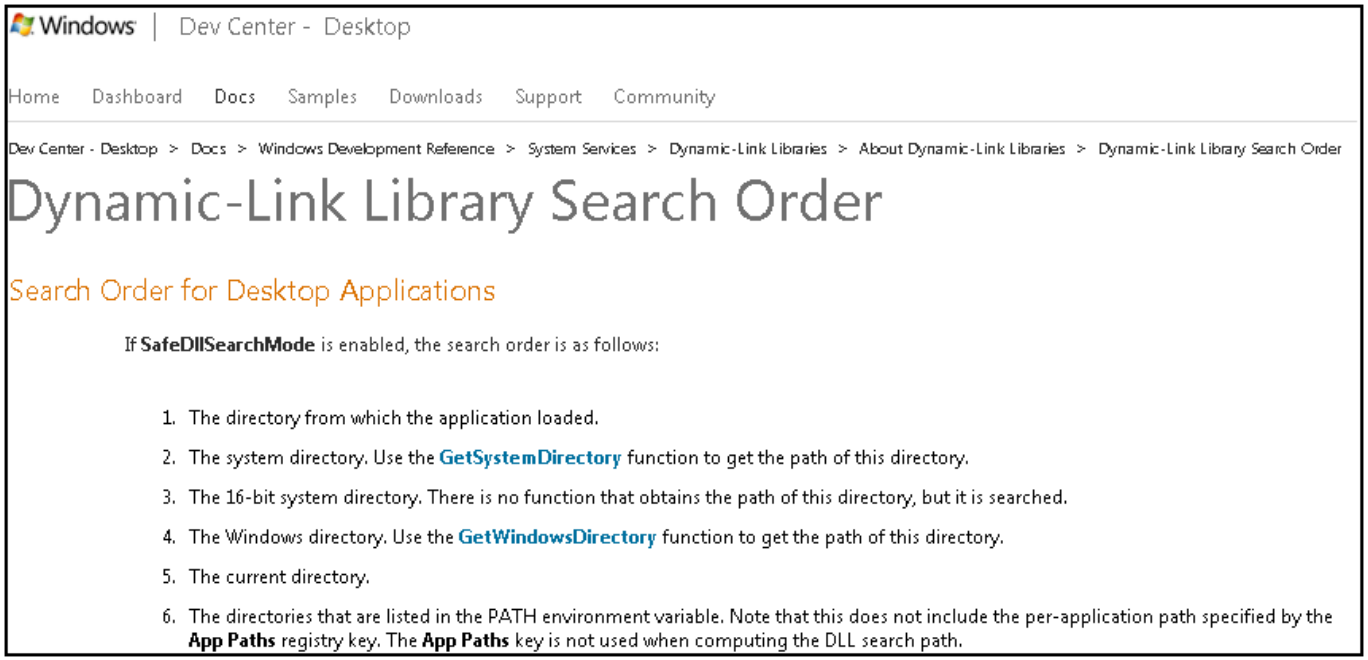
Communication channels

How can we monitor communication channels between HW tokens and their CSPs?

- create fake winscard.dll

  works as a proxy, and creates logs (e.g. about PIN code)
  kind of DLL preloading attack (see Dynamic-Link Library Security topic)

  ... even if „SafeDllSearchMode" registry is present and is set to value „1"!



Windows | Dev Center - Desktop

Home Dashboard Docs Samples Downloads Support Community

Dev Center - Desktop > Docs > Windows Development Reference > System Services > Dynamic-Link Libraries > About Dynamic-Link Libraries > Dynamic-Link Library Search Order

Dynamic-Link Library Search Order

Search Order for Desktop Applications

If **SafeDllSearchMode** is enabled, the search order is as follows:

1. The directory from which the application loaded.
2. The system directory. Use the **GetSystemDirectory** function to get the path of this directory.
3. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
4. The Windows directory. Use the **GetWindowsDirectory** function to get the path of this directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable. Note that this does not include the per-application path specified by the **App Paths** registry key. The **App Paths** key is not used when computing the DLL search path.

source: microsoft.com
(http://msdn.microsoft.com/en-us/library/windows/desktop/ms682586.aspx)

Communication channels

What can be the countermeasures?

- be sure that encrypted APDUs (e.g. PIN codes) are sent to HW tokens

CWA 14890-1

March 2004

Application Interface for smart cards used as Secure Signature Creation Devices
Part 1: Basic requirements

8.2.2 SCA in untrusted environment

A device authentication shall be used if the operating environment of the card cannot be entirely trusted. This can be the case in public signature terminals or other devices, that cannot provide a trusted channel.

Figure 8-4: Communication in untrusted environment

Device authentication is mutual. The ICC shall authenticate the IFD and vice versa. The order of authentication, however, may differ, depending on the implemented scheme.

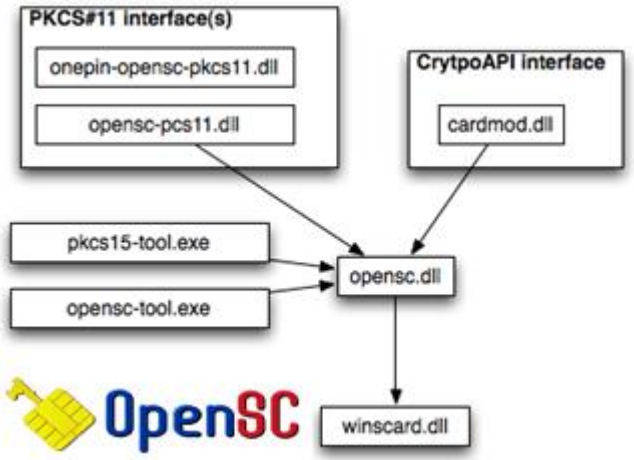
After successful device authentication, session keys are available on both sides to be used in subsequent transmissions. The appropriate secure messaging is in compliance with ISO/IEC 7816-4 [11] and described in 9 "Secure Messaging" on page 9-67.

Examples for an untrusted environment are

- SCA and SSCD are not at the same location, i.e. the card is remote

source: cenorm.be
(ftp://ftp.cenorm.be/Public/Cwas/e-europe/esign/cwa14890-01-2004-Mar.pdf)

e.g. use eID framework or other CSPs that implement also optional parts of CEN CWA 14890 requirements (see „Secure Messaging")

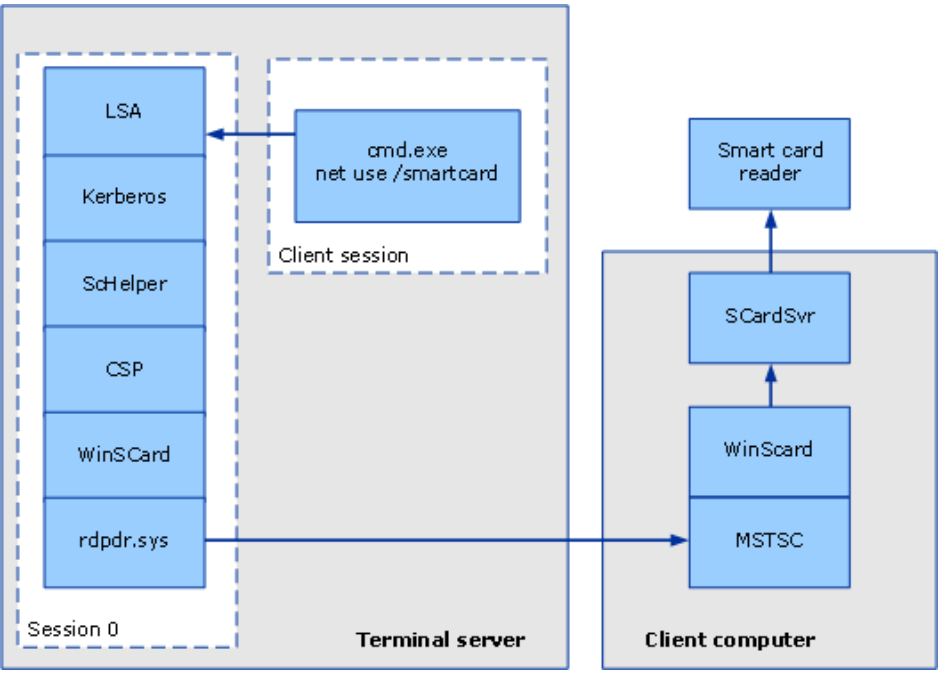


source: opensc-project.org
(http://www.opensc-project.org/opensc/wiki/MiniDriver)

Communication channels

„Let's use HW tokens remotely without any user interaction in the background!"

Communication channels

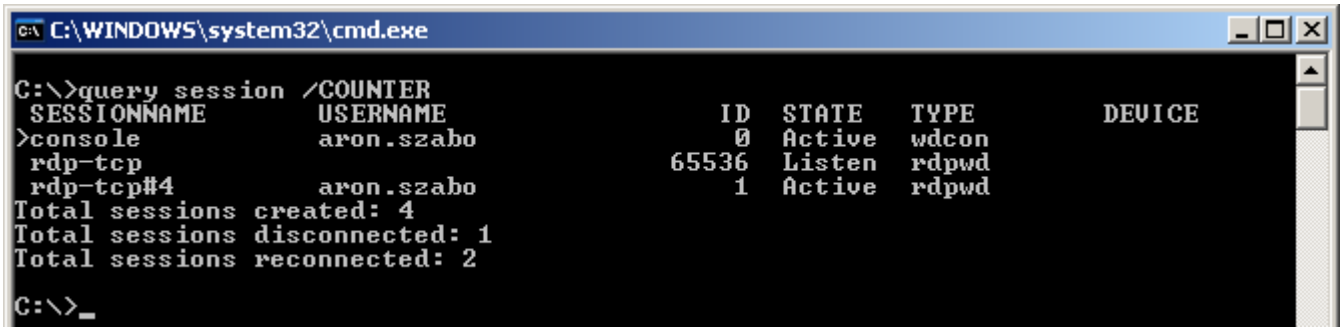


The communication channels between HW tokens and their CSPs should be protected...

The winscard.dll is also important at forwarding communication either to local devices or to remote devices (connected in Terminal Session).

... but we can also replace original winscard.dll on a remote machine, and inject APDUs remotely!

source: microsoft.com
(http://msdn.microsoft.com/en-us/library/bb905527.aspx)



Communication channels

How can we execute HW token commands remotely?

- locate a server which can be accessed

  replace winscard.dll with fake one in order to log all APDU communications

- sleep()

  wait for e.g. system administrator to log in to this attacked server via RDP

- monitor and replay APDUs

  if the „Smart cards" local device was connected via RDP by remote user ...
  if the HW token of remote user was in the reader...
  if the HW token was used during this RDP session by remote user ...

  then we get APDUs (including PIN/password)!
  then we can replay these APDUs whenever this RDP session exists!
  then we can create digital signatures remotely in the background!

Communication channels

What can be the countermeasures?

- if you need to administer another computer remotely, do not use RDP
- if you use RDP, do not connect „Smart cards” local devices
- if you connect „Smart cards” local devices, do not leave your HW token in the reader or in the USB port

Communication channels

Thank you!