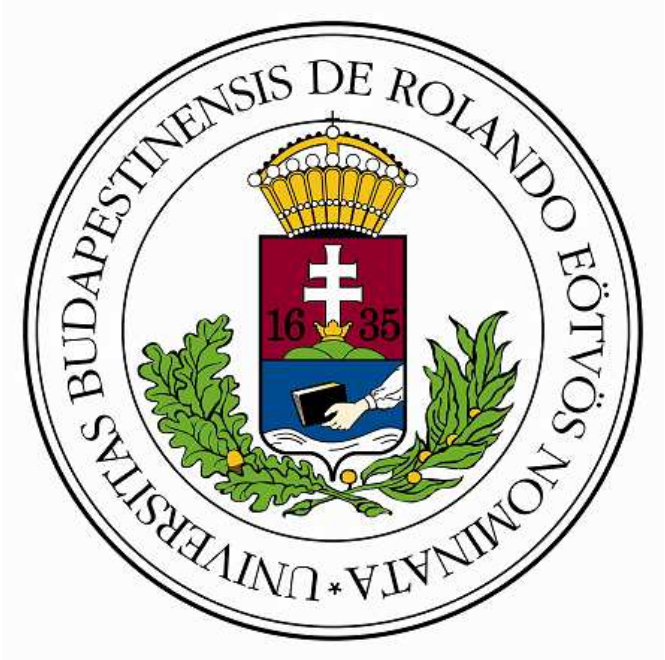
Cryptography:
In general



Áron Szabó

ELTE

2018-11-28

Model anatomy

Models

one-way function vs. two-way function

- one-way function:
  one-way function, fixed-length output, avalanche effect

- two-way function:
  symmetric or asymmetric

symmetric vs. asymmetric (vs. hybrid)

- symmetric:
  one private key (e.g. derived from password/passphrase)

- asymmetric:
  one private key, one public key

Model anatomy

Models

- asymmetric:
  one private key, one public key

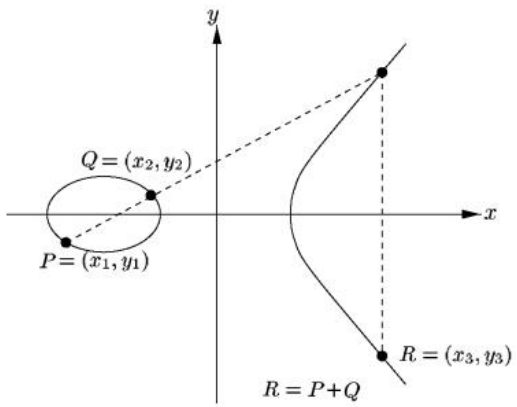
  IFP - Integer Factorization Problem
  (e.g. RSA)

```
p = 13
q = 11
n = p * q = ?
n = 143

n = 437
p = ? ... (23)
q = ? ... (19)
```

  DLP – Discrete Logarithm Problem
  (e.g. DSA)

```
a = 10
b = ?
b = 3^a mod 17 =  8

b = 3^a  mod 17 = 14
a = ? ... (9)
```

  ECDLP – Elliptic Curve Discrete Logarithm Problem
  (e.g. ECDSA)



  difficulty and complexity:
  class NP-intermediate
  (NP-complete is not proven)

Model anatomy

Models

Trusted Third Party vs. Web of Trust

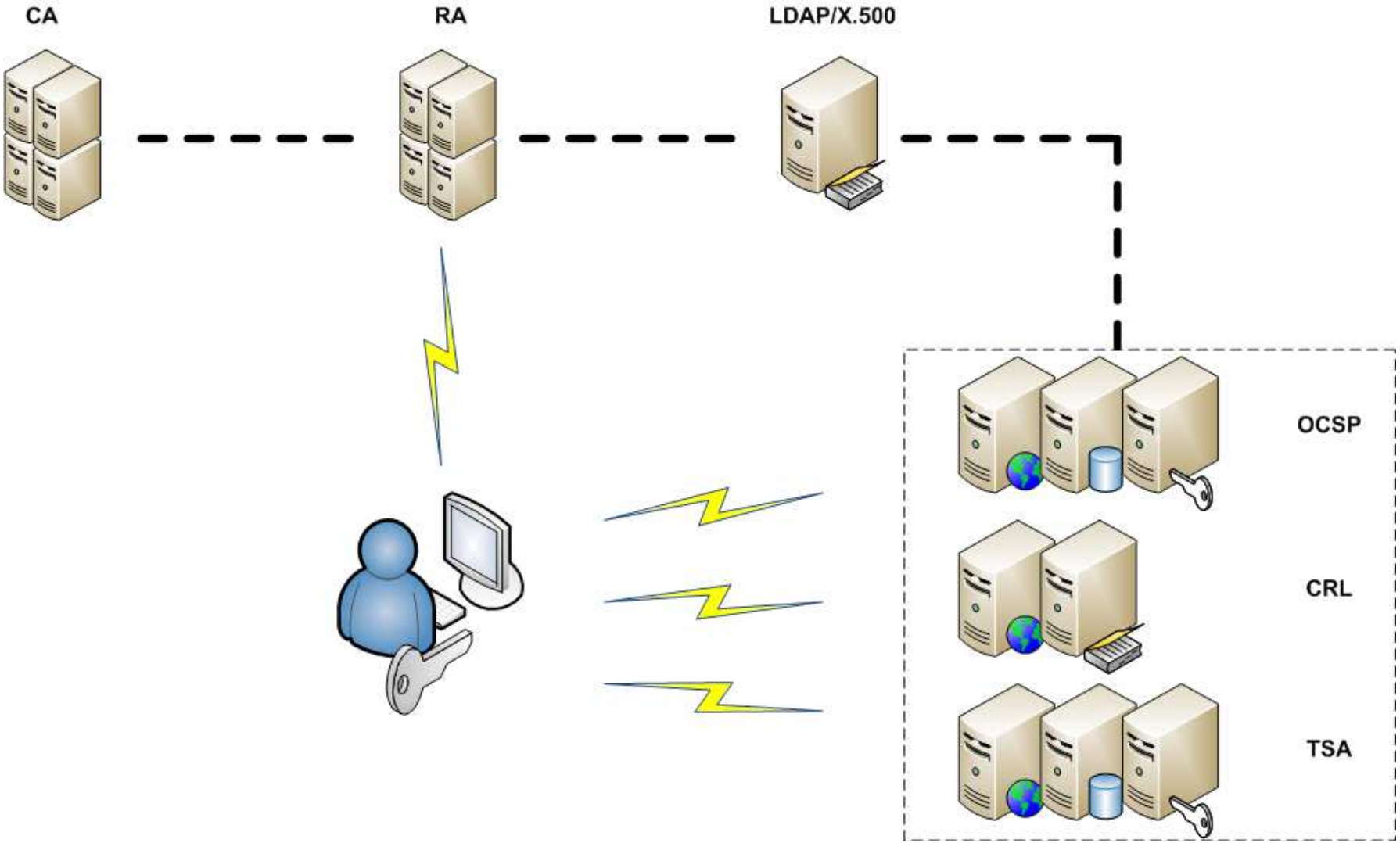
- CA-hierarchy
- PGP, GPG...

X.509 vs. non-X.509

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

CA anatomy

Certificate Authority



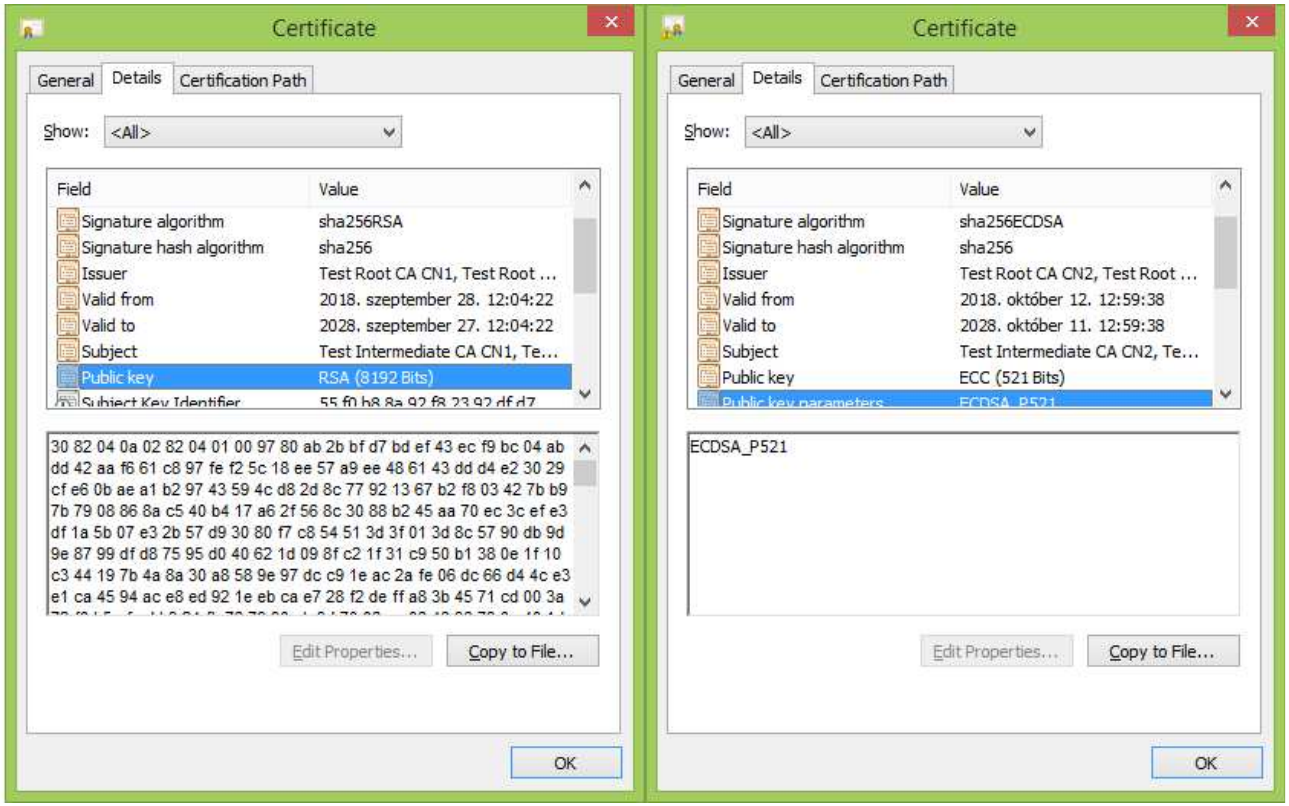
CA anatomy

Certificate Authority

Signed data of Subject
- natural/legal attributes
  and
- public key.

Signed by Issuer
- private key.

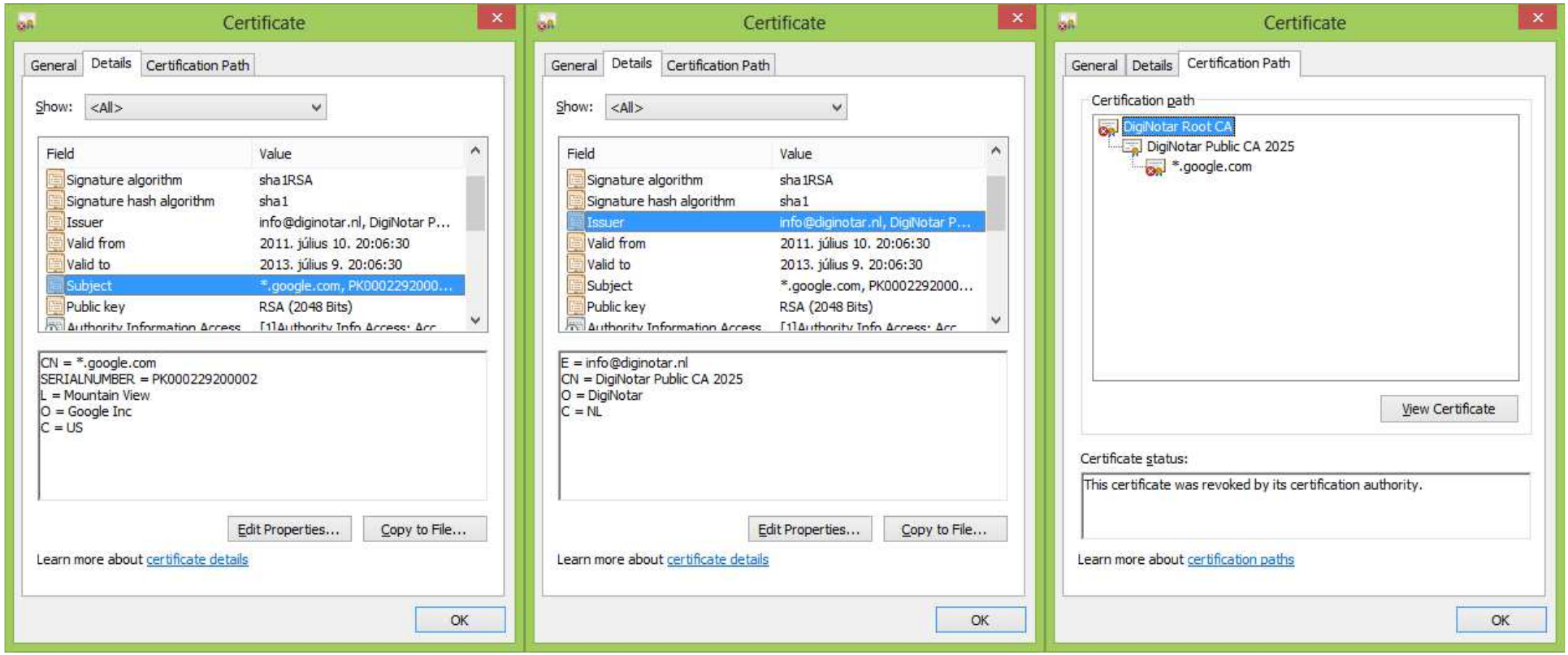
Keystore data can be
- SW (PKCS#12)
  or
- HW (HSM, smart card).



CA anatomy

E-GROUP
SOFTWARE & BEYOND

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"

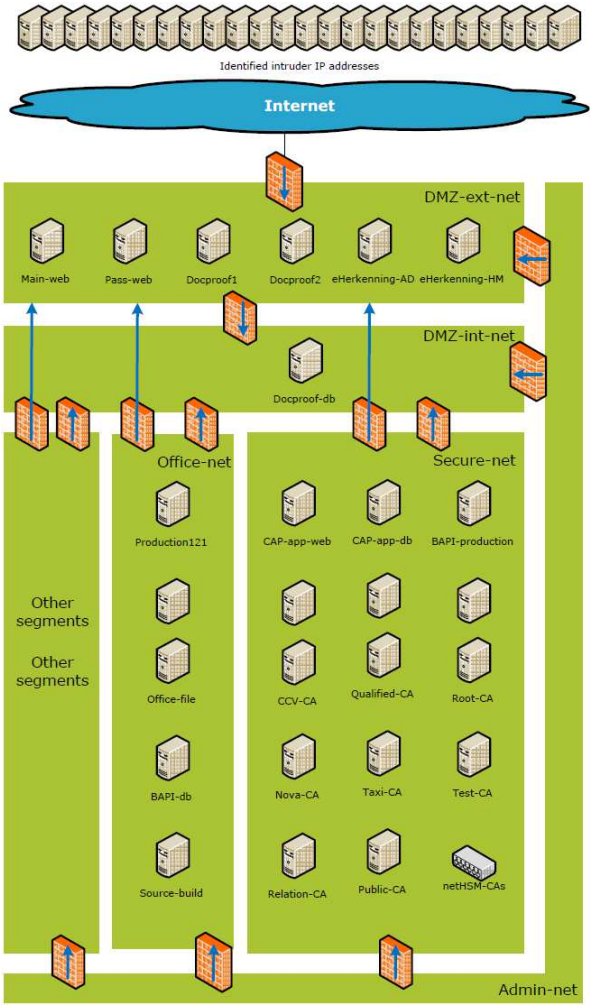
CA software

- "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"

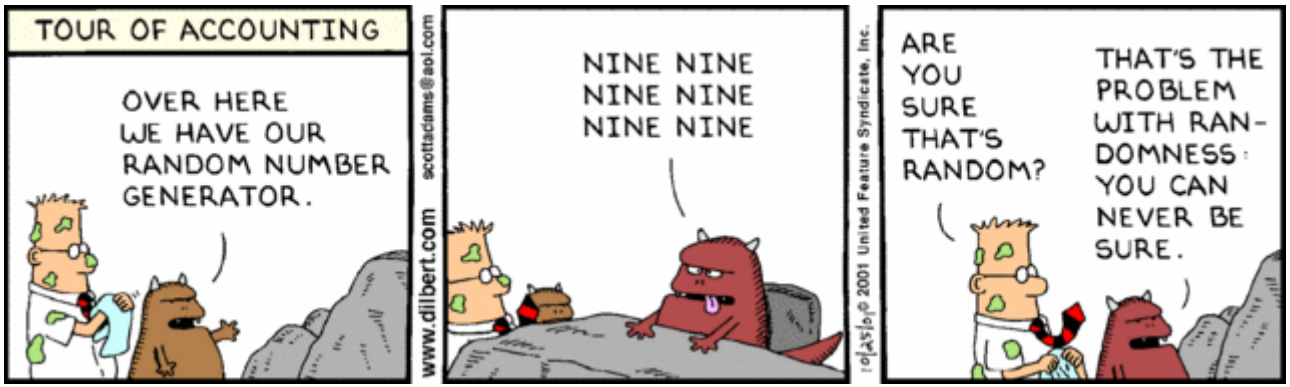
Date	Notes
2011-06-17	"the Main-web and the Docproof2 web server were compromised"
2011-07-01	"first scanning activity occurred in Secure-net"
2011-07-10	"first rogue certificate was successfully created on the Relation-CA server [...]OCSP requests for rogue certificates started arriving [...] from an 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

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 SSL/TLS sites and code signer keys (trusted CAs of Microsoft Windows)
- who generated the vulnerable key pairs and the PKCS#10?

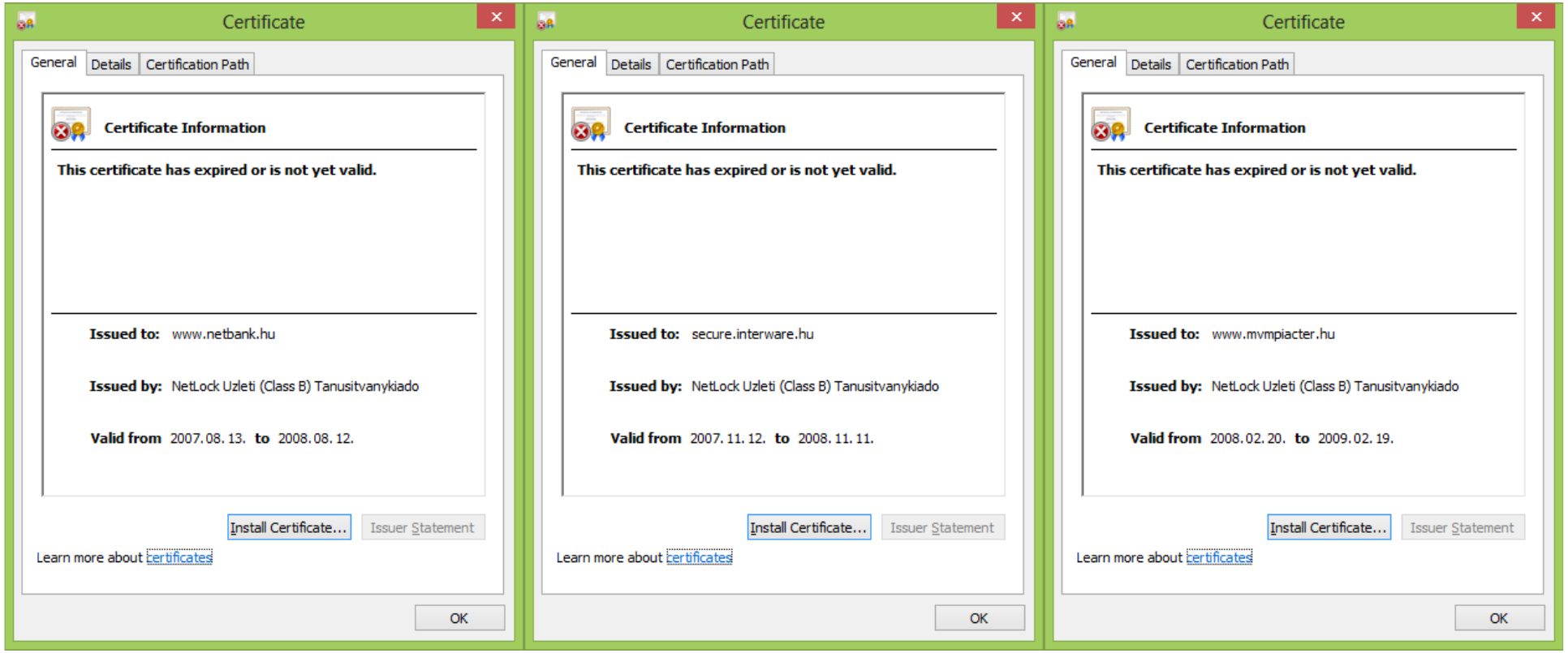


CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and OpenSSL

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

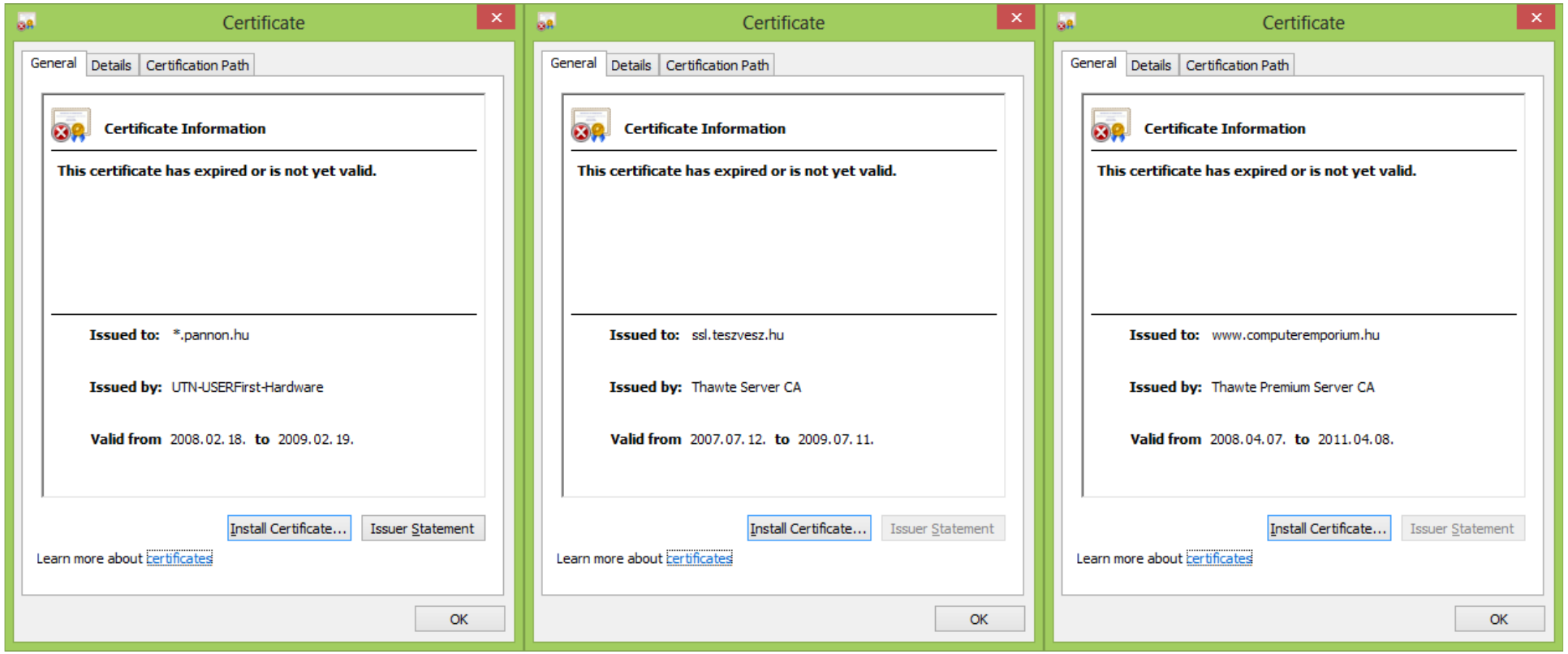


CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and OpenSSL

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



CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and key generation functions

- Infineon RSA library — 2017-10-16, 2017-10-30
  ROCA (Return of Coppersmith's Attack)
- sectors
  - government — Estonia (eID), Slovakia (eID), Spain (eID)
  - bank — Poland (Gemalto IDPrime .NET)
  - other — (e.g. Yubico YubiKey 4, TPM in HP, Lenovo laptop)
- devices — (e.g. NXP?)
- codes — (e.g. BouncyCastle?)
- key sizes — (e.g. RSA 3072 bits? ECC?)
- operations — (e.g. revoked CA, large CRLs, lack of functionality)

Name	Type	Size
svkeidaca_2017-10-30.crl	Certificate Revocation List	176 KB
svkeidaca_2017-10-31.crl	Certificate Revocation List	8 571 KB
svkeidaca_2018-10-16.crl	Certificate Revocation List	8 638 KB
svkeidsca_2017-10-30.crl	Certificate Revocation List	169 KB
svkeidsca_2017-10-31.crl	Certificate Revocation List	8 559 KB
svkeidsca_2018-10-16.crl	Certificate Revocation List	8 616 KB

CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and key generation functions

https://asecuritysite.com/encryption/copper

```
p = k * M + (65537^a mod M)
n = 39
M = 962947420735983927056946215901134429196419130606213075415963491270
k = 8
a = 3
p = 770357936588787141645556972720907543357135304484970488481556973931 3

q = k * M + (65537^a mod M)
n = 39
M = 962947420735983927056946215901134429196419130606213075415963491270
k = 8
a = 1
q = 770357936588787141645556972720907543357135304484970460332770799569 7
```

https://www.mobilefish.com/services/rsa_key_generation/rsa_key_generation.php

```
phi(n) = (p - 1) * (q - 1)
phi(n) = 14e6e6e84a6c9bc76fcd3e57153c4eb8069b4a417024e38cc40c7cad483421d8d46903b67e137fe19997b0afcf43d17967663b7757574900

n = p * q (modulus)
n = 14e6e6e84a6c9bc76fcd3e57153c4eb8069b4a417024e38cc40c7cadda80dc43be06fdbec34b7aadee2c3ea0918171be4031f98934d10561
    (decimal: 59345135046533379070949212431851844835352661985729183720911732659737228034511329429220576921531616803463218368738626766508388015736161)

e = (publicExponent)
e = 10001
    (decimal: 65537)

d = e^-1 mod phi (privateExponent)
d = 10764b26687761b991b47484e194bdeaf2491c12ade05fd75e10f7a1025d0ccb597fdc0cbeff5c98b1ff66891114b8c685495f4c7edda001

p = (prime1)
p = 49265d3574cefd04229bfd662a4a46f8611ed0226c665f0a6ebdde31
    (decimal: 7703579365887871416455569727209075433571353044849704884815569739313)
    (n = 39, k = 8, a = 3)

q = (prime2)
q = 49265d3574cefd04229bfd662a4a46f8611ed0226c655f076ebbde31
    (decimal: 7703579365887871416455569727209075433571353044849704603327707995697)
    (n = 39, k = 8, a = 1)

dP = d mod (p - 1) (exponent1)
dP = 2646c9dd2ffb23902760028f942f7bc57a647a3a990854510c3393f1

dQ = d mod (q - 1) (exponent2)
dQ = 2646c9dd2ffb23902760028f942f7bc57a647a3a9907ce5b004993f1

qInv = q^-1 mod p (coefficient)
qInv = 23c7b4c478c1d3174522a8362a1536f33b637a1bfc32fef67f732fe0

public_key_vulnerable_generated.txt (base64 ASN.1)
MFQwDQYJKoZIhvcNAQEBBQADQwAwQAI5ABTm5uhKbJvHb80+VxU8TrgGm0pBcCTjjMQMfK3agNxDvgb9vsNLeq3uLD6gkYFxvkAx+Yk00QVhAgMBAAE=
```

CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and key generation functions

KeyChest - Profile

https://keychest.net/roca

ROCA vulnerability summary

Paste key | File upload | Send e-mail | GitHub account | PGP/GnuPG keyring | Offline test

Paste your key

If you can copy your **public key** into clipboard, you can test it by pasting it to the text box. Please check the public key is in one of the supported formats as listed below.

```
-----BEGIN CERTIFICATE-----
MIIFujCCA6KgAwIBAgIIRc5294wMI4wDQYJKoZIhvcNAQELBQAwWzELMAkGA1UE
[redacted]
3v0r91kKEnqv2//Byi6svD9OSz651GU5Hs6DlhZtswSd50KWo7I6WRkbDITIQFWU
```

Supported formats are:

- X509 Certificate, PEM encoded (include ASCII armor "-----BEGIN"); example
- SSH public RSA key (starting with "ssh-rsa "); example
- RSA PEM encoded public key (include ASCII armor "-----BEGIN"); example
- PGP public key; example
- Raw RSA modulus (hex / decimal / base64 encoded); and
- S/MIME PKCS7 Signature.

Test key

CA anatomy

E-GROUP
SOFTWARE & BEYOND

code review and key generation functions

KeyChest - Profile

ROCA vulnerability summa

Paste key | File upload

Paste your key

If you can copy your **public key** int
below.

-----BEGIN CERTIFICATE-----
MIIFujCCA6KgAwIBAgIIRc5294w
[redacted]
3v0r91kKEnqv2//Bvi6svD9OSz65

Supported formats are:

- X509 Certificate, PEM encod
- SSH public RSA key (starting
- RSA PEM encoded public ke
- PGP public key; example
- Raw RSA modulus (hex / dec
- S/MIME PKCS7 Signature.

KeyChest - Profile

https://keychest.net/roca

Search

Error during the key processing, please refresh the page and try again.

Results

The key is subject to ROCA, which reduces the security bellow generally acceptable level.

Type / Interpretation	X509 certificate (PEM)
Fingerprint	17882ba91244e5ea3ff2bbeb9f742e326d72732d3e19680880da2d537c0971c1
Subject	CN: [redacted]
Issuer	C: [redacted]
Created on	Aug 2nd 2016
Expiring on	Aug 2nd 2018
Bit length	2048
Test result	Subject to ROCA vulnerability, insecure.

Security notice: Keys with the ROCA vulnerability provide significantly lower level of security than expected for RSA keys of the given length. Addendum: The BSI, German security evaluation agency, has evaluated the strength of RSA keys for impacted cards Atos CardOS V5.0 and concluded that 'the key length 3,072 and 3,584 have a security level above 100 bit', the required security level for qualified signatures. [link to BSI].

We are Enigma Bridge Ltd, 20 Bridge St, Cambridge, CB2 1UF, United Kingdom and we read keychest@enigmabridge.com

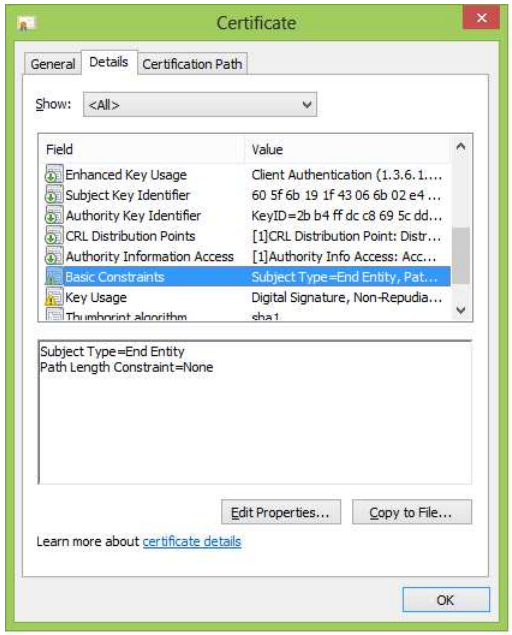
Terms of Service | Privacy Policy

ENIGMA BRIDGE

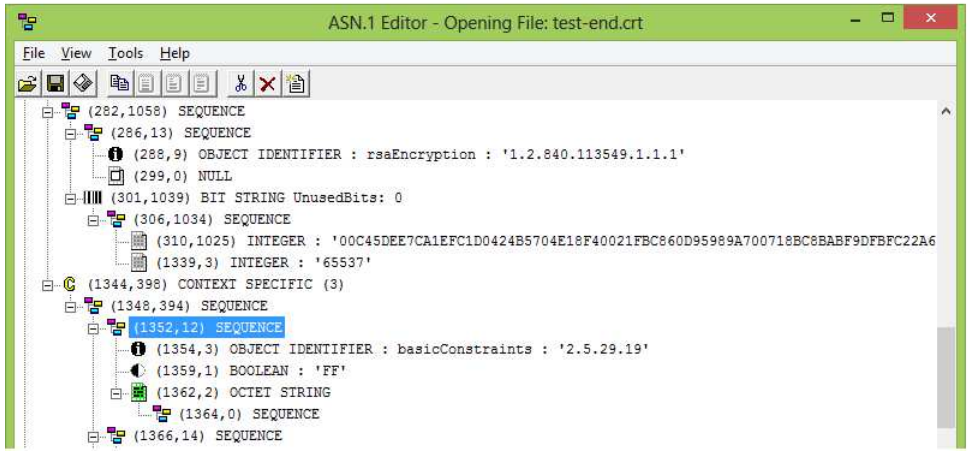
X.509 anatomy

E-GROUP
SOFTWARE & BEYOND

basicConstraints



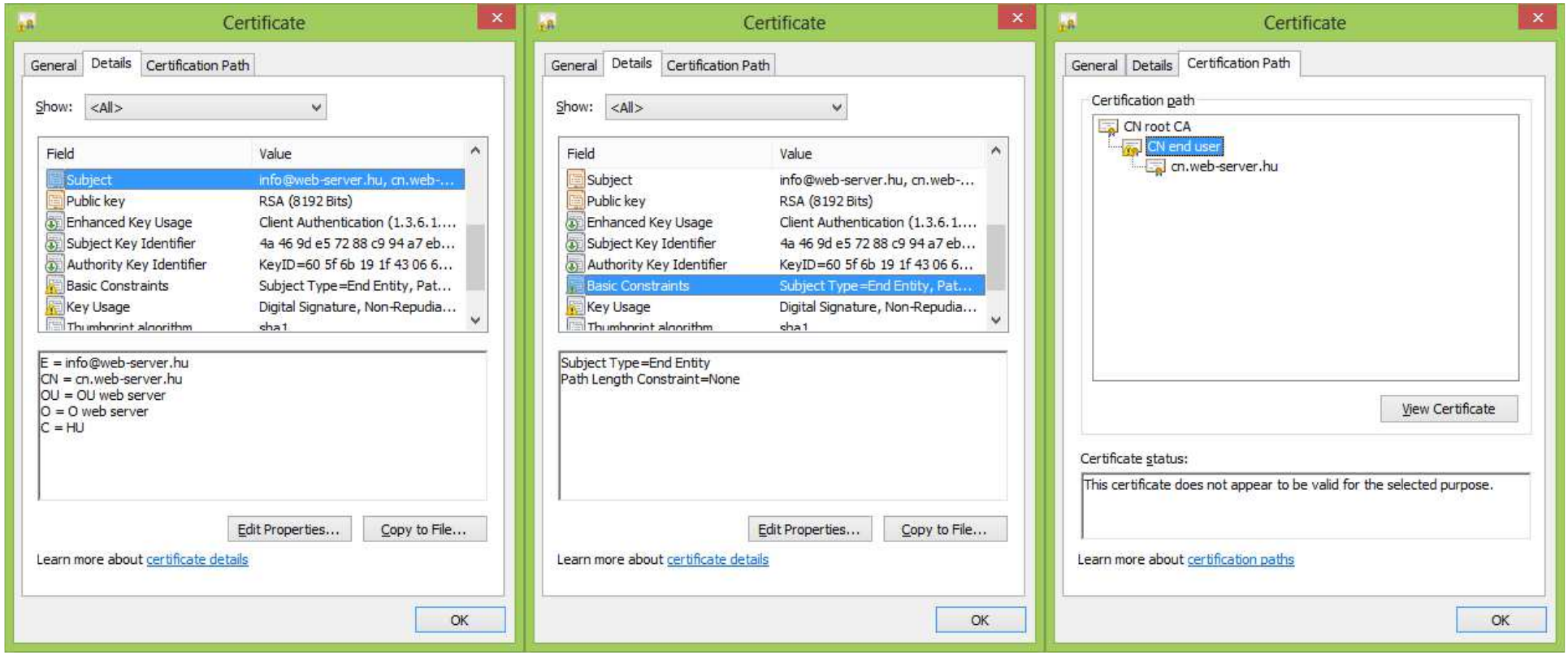

```
basicConstraints

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

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



X.509 anatomy

E-GROUP
SOFTWARE & BEYOND

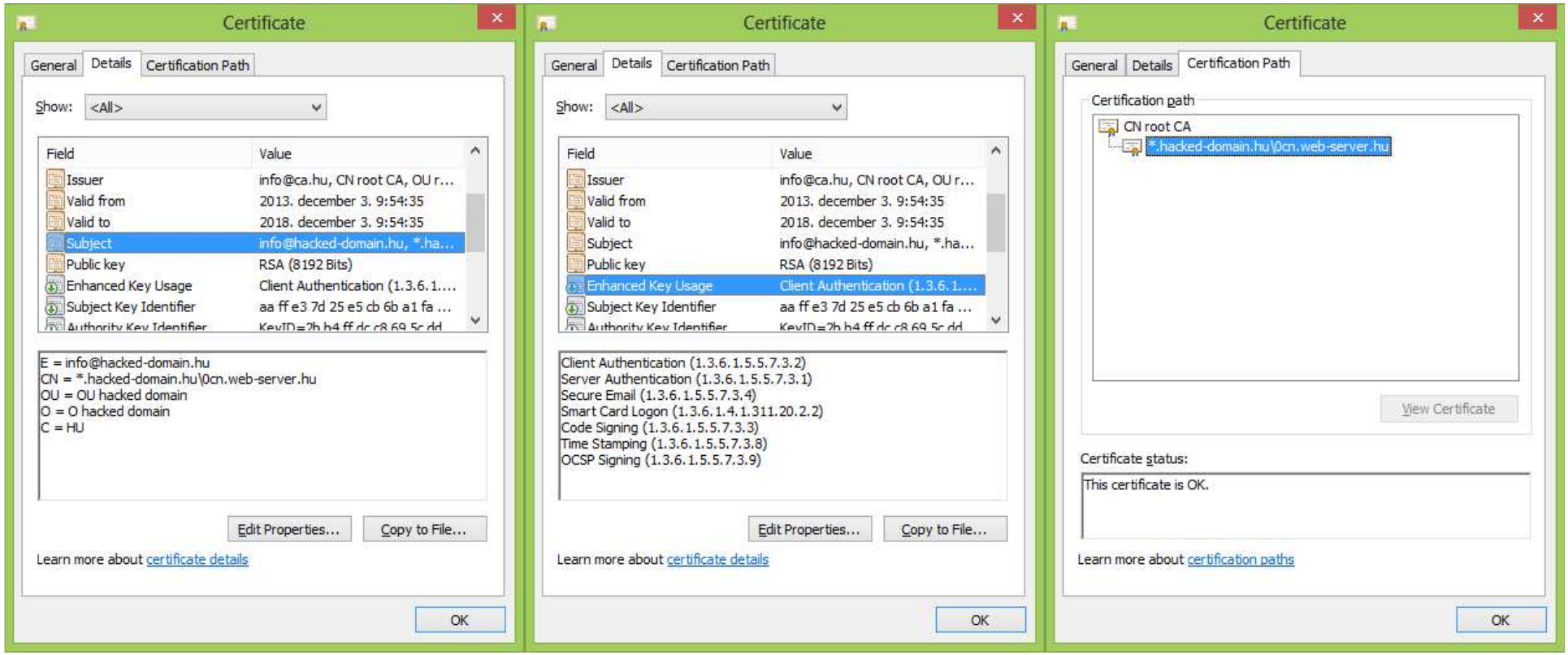
basicConstraints and Moxie Marlinspike



X.509 anatomy

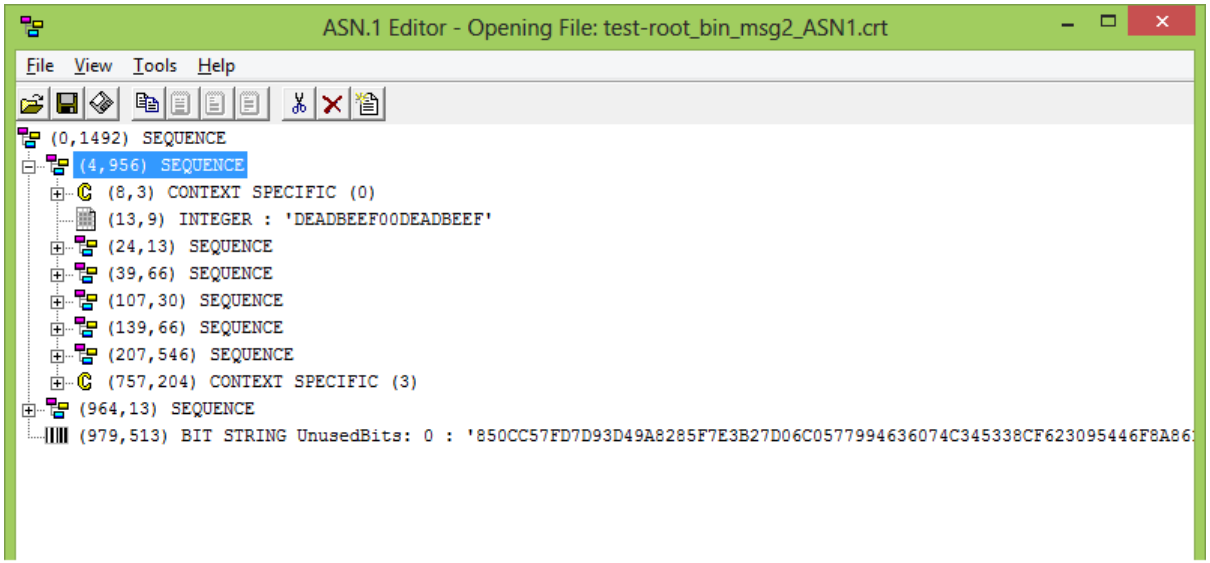
E-GROUP
SOFTWARE & BEYOND

subject „\0” and Moxie Marlinspike



Algorithm anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



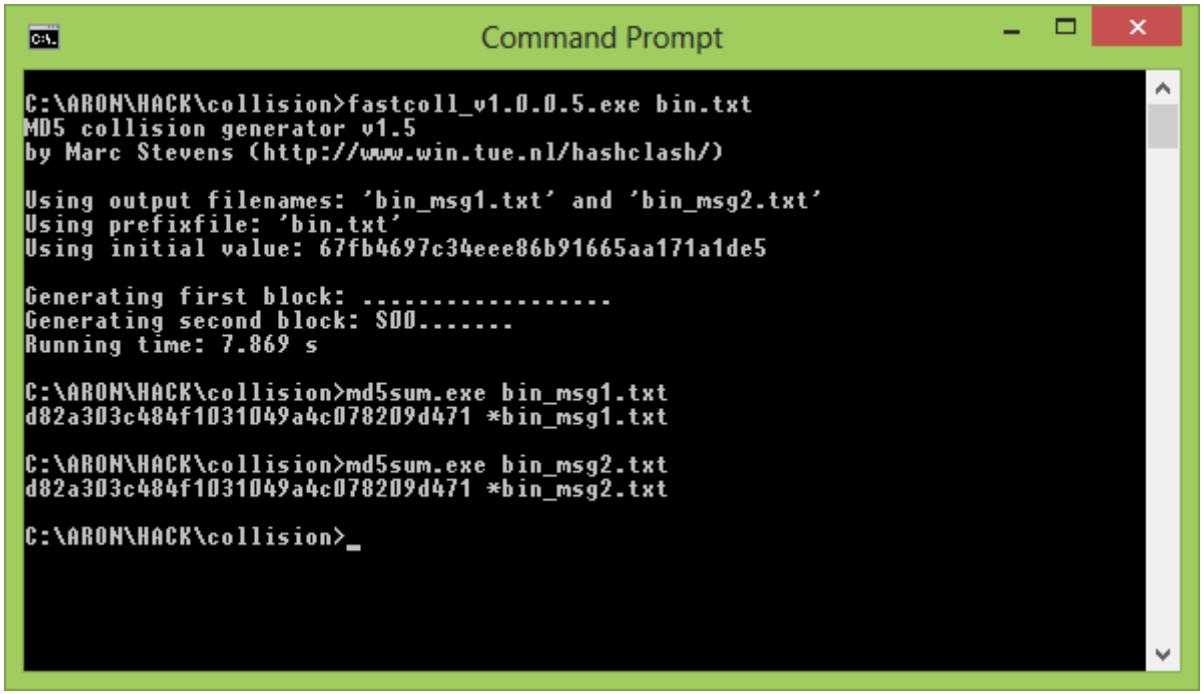
Certificate and TBSCertificate

```
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
                                 }
```

Algorithm anatomy

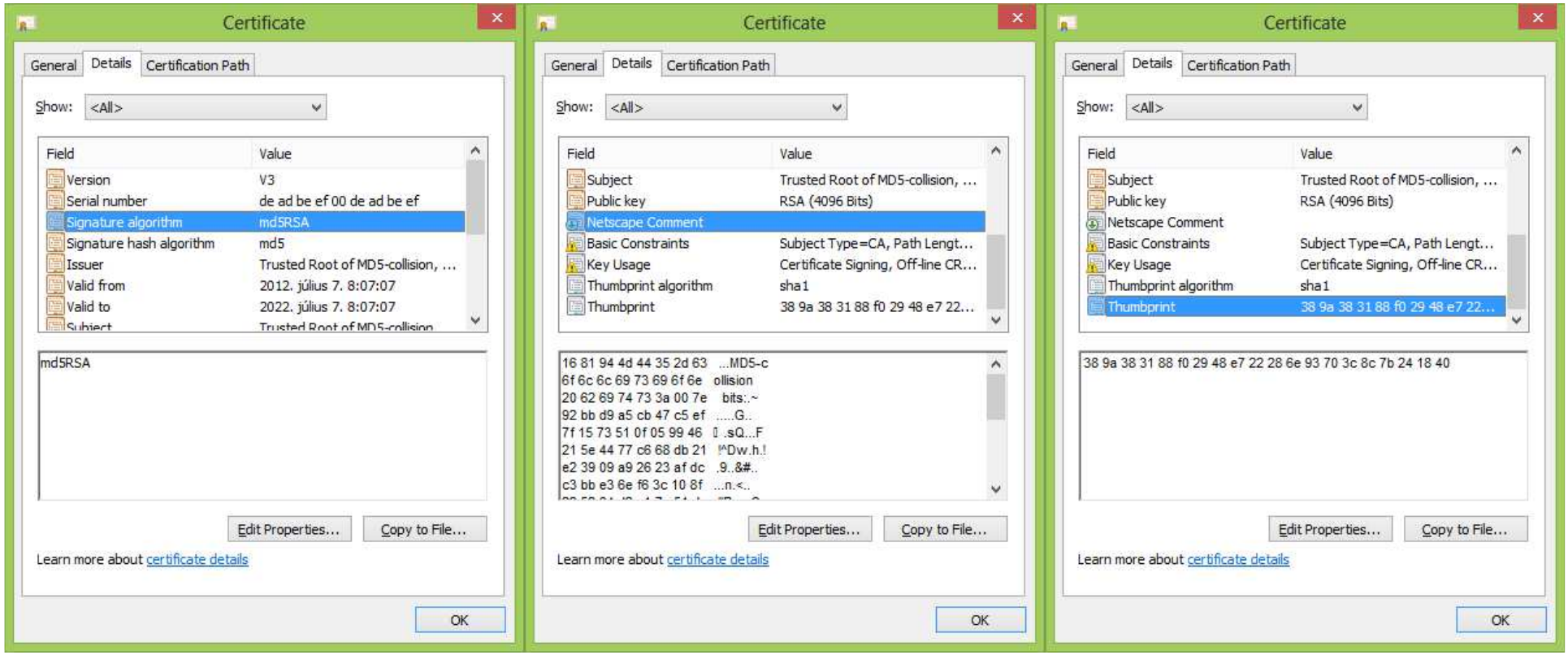
MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima

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



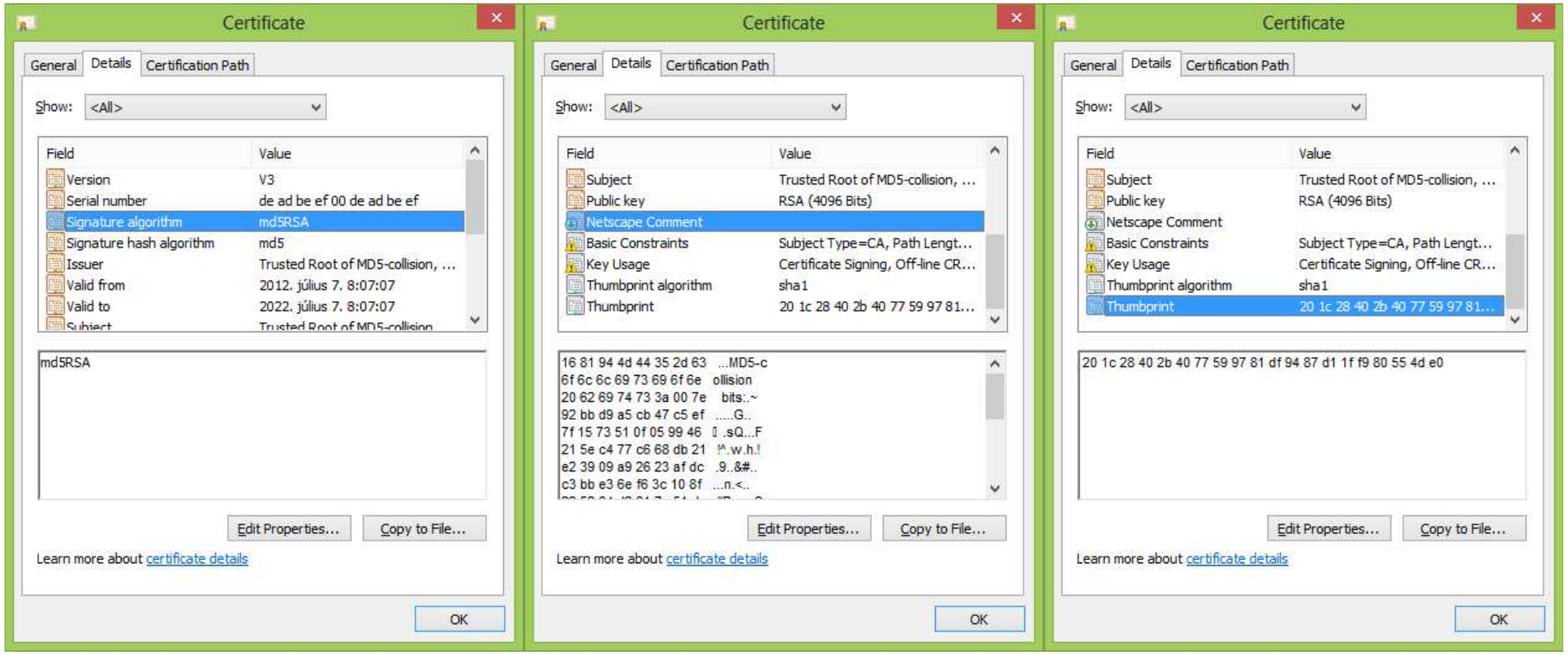
Algorithm anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



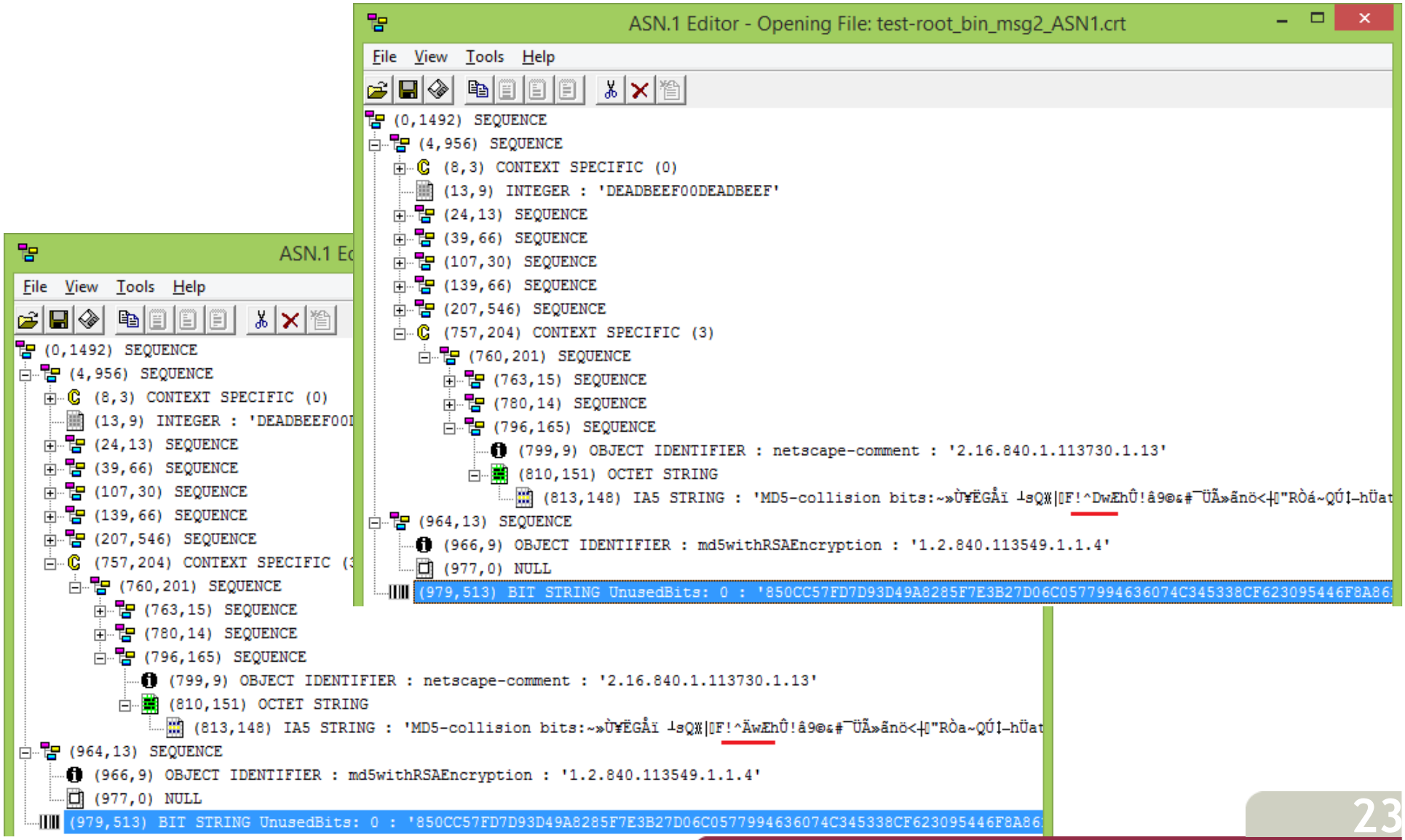
Algorithm anatomy

MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



Algorithm anatomy

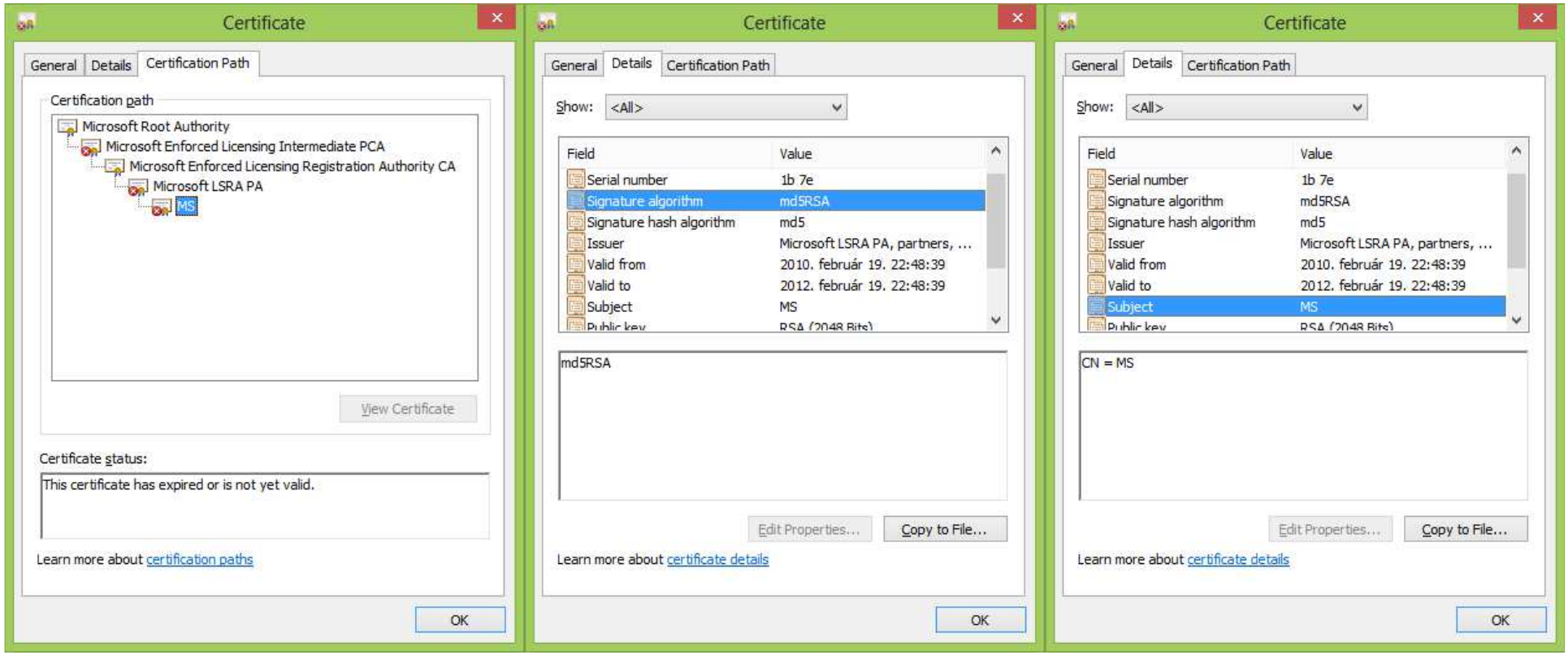
MD5 collision and Xiaoyun Wang, Marc Stevens, Vlastimil Klima



Algorithm anatomy

E-GROUP
SOFTWARE & BEYOND

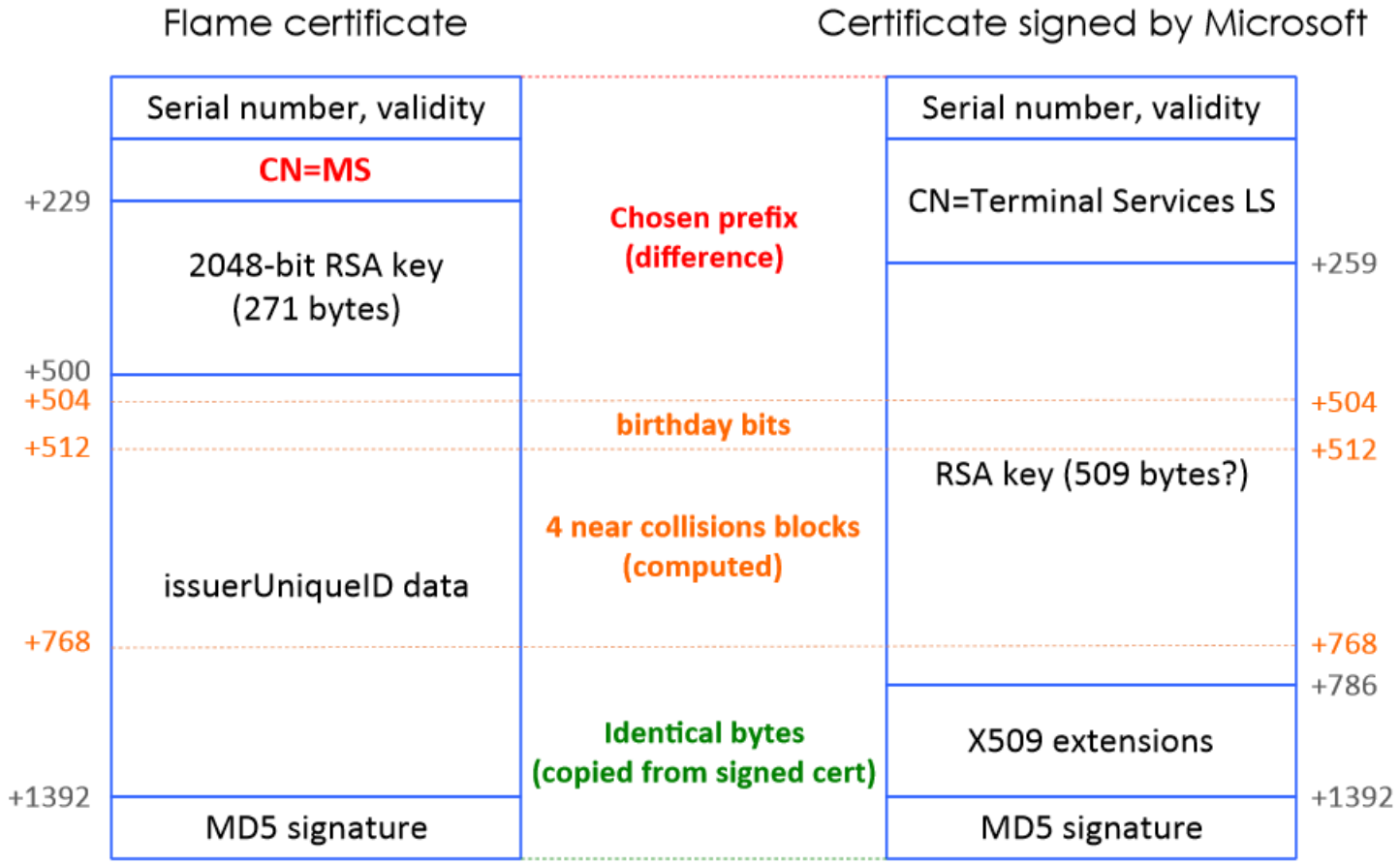
MD5 collision and Flame, WuSetupV.exe



Algorithm anatomy

MD5 collision and Flame, WuSetupV.exe

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



Algorithm anatomy

SHA-1 collision and E. A. Grechnikov

- collisions for 75-step reduced SHA-1
- using SHA-2 is recommended
- NIST selected Keccak (2012-10-02) as the basis for SHA-3

http://localhost/collision/index.php Collisions for 75-step reduc...

74	e282e133a7d1b8e51f42f50e6112456f962f8c2f	d8adb231b3ee98c26da9c351a4a4218d81338aca
75	**9250b597f77d85324fd5520a8d0fcf9223be148c**	**9250b597f77d85324fd5520a8d0fcf9223be148c**
76	2021fb9a50ad6c3a031d5760dbfe3f311393a8cc	9a4d2b687d7d6fdaa8684cfb84d08fc252ec9e7a
77	a5b3b3d32c9c59cd7ff715865cabf6a5fcc8c478	bdae2499eb3617929d5d92b64f6c146c87b4dee5
78	ad57dbad9114c267eb39bab7ec42e1b277487216	21f3531533c2bf2d4b6c8a06ed70054afd9f1071
79	808a863614ad183099c65805389fee414ccf0826	18b5c5eb0dc7f6cd63c9145402c0012add4937f9
80	**7e91244743b36fdbc9c1273b2c2cf0ef314b4ad4**	**8366a31cb05717f68b82093de9579c76a02f56de**
sha1()	**7e91244743b36fdbc9c1273b2c2cf0ef314b4ad4**	**8366a31cb05717f68b82093de9579c76a02f56de**

red: SHA-1 collision is found
green: final (80-step) value of third-party SHA-1 implementation - sha1_80rounds()
black: final (80-step) value of official PHP SHA-1 implementation - sha1()

Algorithm anatomy

SHA-1 collision and Malicious SHA-1 project

- collisions for 80-step (full) SHA-1 with modified constant words K(t)
- K(t) = [5A827999, 6ED9EBA1, 8F1BBCDC, CA62C1D6]

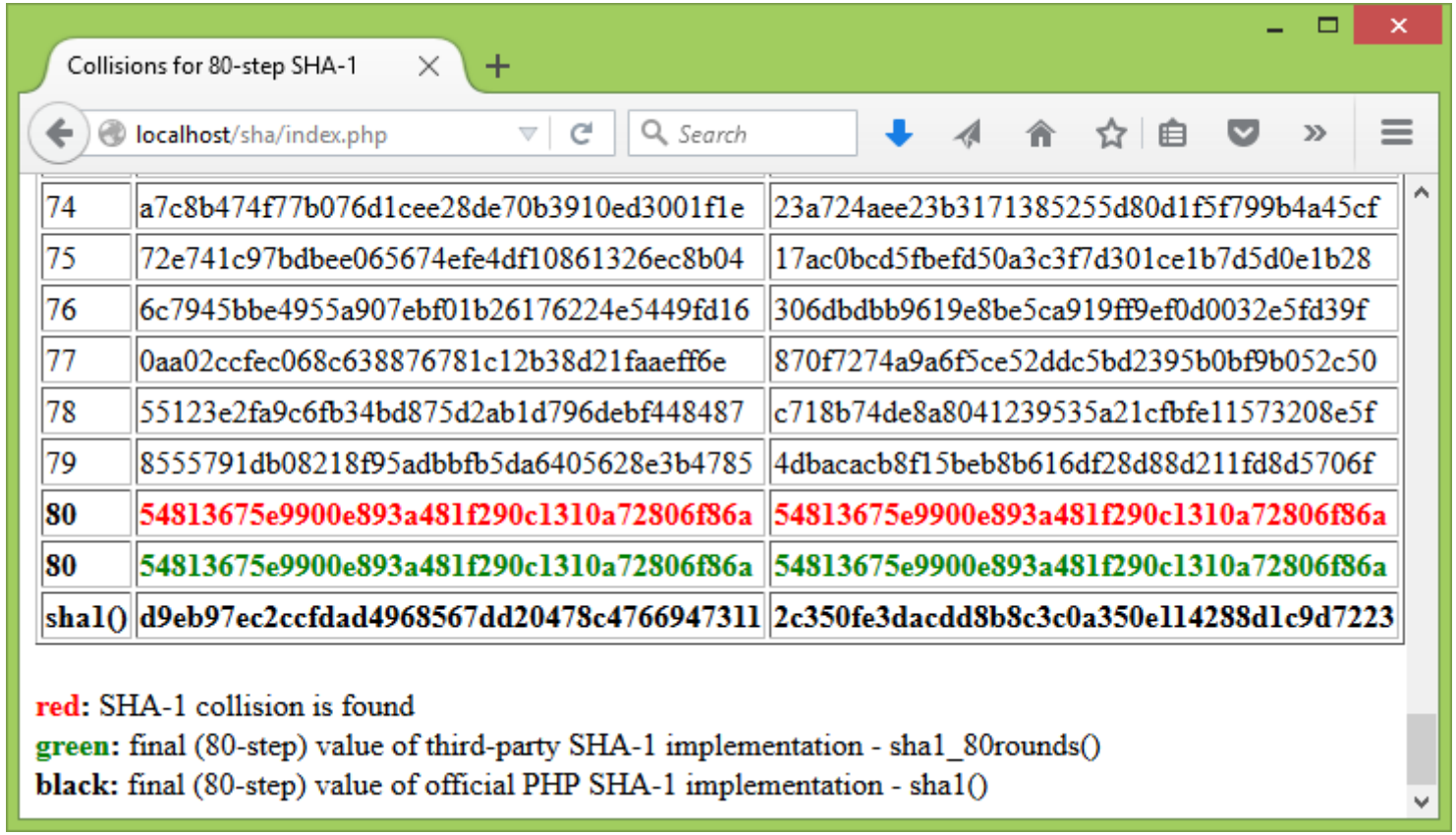
Command Prompt

```
C:\ARON\HACK\collision\SHA>python.exe sha1mod.py
Calculates the SHA-1 sum of a given file (optionally with custom constants)
Usage:
        ./sha1mod.py <filename> [<K0> <K1> <K2> <K3>]
Examples:
        ./sha1mod.py file0.bin
        ./sha1mod.py file0.bin 5A827999 6ED9EBA1 8F1BBCDC CA62C1D6

C:\ARON\HACK\collision\SHA>python.exe sha1mod.py eve1.sh 5A827999 6ED9EBA1 8F1BB
CDC CA62C1D6
FCB2E7BF 18265B08 FA364E80 28A296BB 9C55B79A

C:\ARON\HACK\collision\SHA>python.exe sha1mod.py eve2.sh 5A827999 6ED9EBA1 8F1BB
CDC CA62C1D6
1F362D92 21356C48 092122FF 551624A4 D2263892

C:\ARON\HACK\collision\SHA>python.exe sha1mod.py eve1.sh 5A827999 88E8EA68 57805
9DE 54324A39
96ED59BE 04518A27 C30F17DE 6F0037F9 B3C3257E

C:\ARON\HACK\collision\SHA>python.exe sha1mod.py eve2.sh 5A827999 88E8EA68 57805
9DE 54324A39
96ED59BE 04518A27 C30F17DE 6F0037F9 B3C3257E

C:\ARON\HACK\collision\SHA>_
```

Algorithm anatomy

SHA-1 collision and Marc Stevens, Pierre Karpman, Thomas Peyrin

- collisions for 80-step (full) SHA-1 with modified initialization vectors H0..H4
- H0..H4 = [506B0178, FF6D1890, 202291FD, 3ADE3871, B2C665EA]



74	a7c8b474f77b076d1cee28de70b3910ed3001f1e	23a724aee23b3171385255d80d1f5f799b4a45cf
75	72e741c97bdbee065674efe4df10861326ec8b04	17ac0bcd5fbefd50a3c3f7d301ce1b7d5d0e1b28
76	6c7945bbe4955a907ebf01b26176224e5449fd16	306dbdbb9619e8be5ca919ff9ef0d0032e5fd39f
77	0aa02ccfec068c638876781c12b38d21faaeff6e	870f7274a9a6f5ce52ddc5bd2395b0bf9b052c50
78	55123e2fa9c6fb34bd875d2ab1d796debf448487	c718b74de8a8041239535a21cfbfe11573208e5f
79	8555791db08218f95adbbfb5da6405628e3b4785	4dbacacb8f15beb8b616df28d88d211fd8d5706f
80	**54813675e9900e893a481f290c1310a72806f86a**	**54813675e9900e893a481f290c1310a72806f86a**
80	**54813675e9900e893a481f290c1310a72806f86a**	**54813675e9900e893a481f290c1310a72806f86a**
sha1()	**d9eb97ec2ccfdad4968567dd20478c4766947311**	**2c350fe3dacdd8b8c3c0a350e114288d1c9d7223**

red: SHA-1 collision is found
green: final (80-step) value of third-party SHA-1 implementation - sha1_80rounds()
black: final (80-step) value of official PHP SHA-1 implementation - sha1()

Algorithm anatomy

SHA-1 forging based on MD5 forging and Thai Duong, Juliano Rizzo

- forging hashes of 80-step (full) SHA-1 with modified initialization vectors H0..H4
- H0..H4 = [67452301, EFCDAB89, 98BADCFE, 10325476, C3D2E1F0]
- length extension attack: Flickr, Scribd, Vimeo...

```
transaction parameters (original)

        uid=User1234&txid=12345678&event=moneytransfer&from=User1234&to=User8888&amount=1000

to-be-hashed data (original)

        pwd=Pass1234&
        uid=User1234&txid=12345678&event=moneytransfer&from=User1234&to=User8888&amount=1000

transaction posted parameters (original)

        POST /submit.php HTTP/1.1
        Host: www.mybank.com
        Content-Type: application/x-www-form-urlencoded; charset=UTF-8

        uid=User1234&txid=12345678&event=moneytransfer&from=User1234&to=User8888&amount=1000
        &hash=487f551ef831a9a24594f286b32bb7fac4281c36
```

Algorithm anatomy

E-GROUP
SOFTWARE & BEYOND

SHA-1 forging based on MD5 forging and Thai Duong, Juliano Rizzo

- inject data, apply padding, modify initialization vectors, continue hashing

```
transaction parameters (injected)

        &from=User1234&to=User6666&amount=1000

to-be-hashed data (injected)

        uid=User1234&txid=12345678&event=moneytransfer&from=User1234&to=User8888&amount=1000
        <padding>
        &from=User1234&to=User6666&amount=1000

transaction posted parameters (injected)

        POST /submit.php HTTP/1.1
        Host: www.mybank.com
        Content-Type: application/x-www-form-urlencoded; charset=UTF-8

        uid=User1234&txid=12345678&event=moneytransfer&from=User1234&to=User8888&amount=1000
        <padding>
        &from=User1234&to=User6666&amount=1000
        &hash=d79c913e23d22609565e28a0ff920250d18f9a72
```

Algorithm anatomy

SHA-1 forging based on MD5 forging and Thai Duong, Juliano Rizzo

Command Prompt

```
C:\ARON\HACK\collision\SHA>python.exe sha1.py "pwd=Pass1234&uid=User1234&txid=12
345678&event=moneytransfer&from=User1234&to=User8888&amount=1000"
msg: 7077643d506173733132333426756964 3d5573657231323334267478696 43d3132333435363
738266576656e743d6d6f6e65797472616e736665722666726f6d3d5573657231323334267 46f3d5
5736572383838382661 6d6f756e743d313030308000000000000000000000000000000000000000000
000000000000000000308
h0: 0x67452301
h1: 0xefcdab89L
h2: 0x98badcfeL
h3: 0x10325476
h4: 0xc3d2e1f0L
487f551ef831a9a24594f286b32bb7fac4281c36

C:\ARON\HACK\collision\SHA>python.exe sha1_len_ext_attack.py 487f551ef831a9a2459
4f286b32bb7fac4281c36 97 "&from=User1234&to=User6666&amount=1000"
msg: 2666726f6d3d5573657231323334267 46f3d5573657236363626616d6f756e743d3130303
08000000000000000000000000000000000000000000000000000000000530
h0: 0x487f551e
h1: 0xf831a9a2L
h2: 0x4594f286
h3: 0xb32bb7faL
h4: 0xc4281c36L
d79c913e23d22609565e28a0ff920250d18f9a72

C:\ARON\HACK\collision\SHA>python.exe sha1.py -xfull 7077643d506173733132333426 7
569643d5573657231323334267478696 43d3132333435363738266576656e743d6d6f6e657974726
16e736665722666726f6d3d5573657231323334267 46f3d5573657238383838266 16d6f756e743d3
130303080000000000000000000000000000000000000000000000000000000000000000 3082666726f6d3
d5573657231323334267 46f3d5573657236363626616d6f756e743d313030308000000000000000
000000000000000000000000000000000000530
msg: 7077643d506173733132333426756964 3d5573657231323334267478696 43d3132333435363
738266576656e743d6d6f6e65797472616e736665722666726f6d3d5573657231323334267 46f3d5
5736572383838382661 6d6f756e743d313030308000000000000000000000000000000000000000000
000000000000000000 3082666726f6d3d5573657231323334267 46f3d5573657236363626616d6
f756e743d313030308000000000000000000000000000000000000000000000000000000000000530
h0: 0x67452301
h1: 0xefcdab89L
h2: 0x98badcfeL
h3: 0x10325476
h4: 0xc3d2e1f0L
d79c913e23d22609565e28a0ff920250d18f9a72

C:\ARON\HACK\collision\SHA>_
```

Algorithm anatomy

SHA-1 collision and Marc Stevens

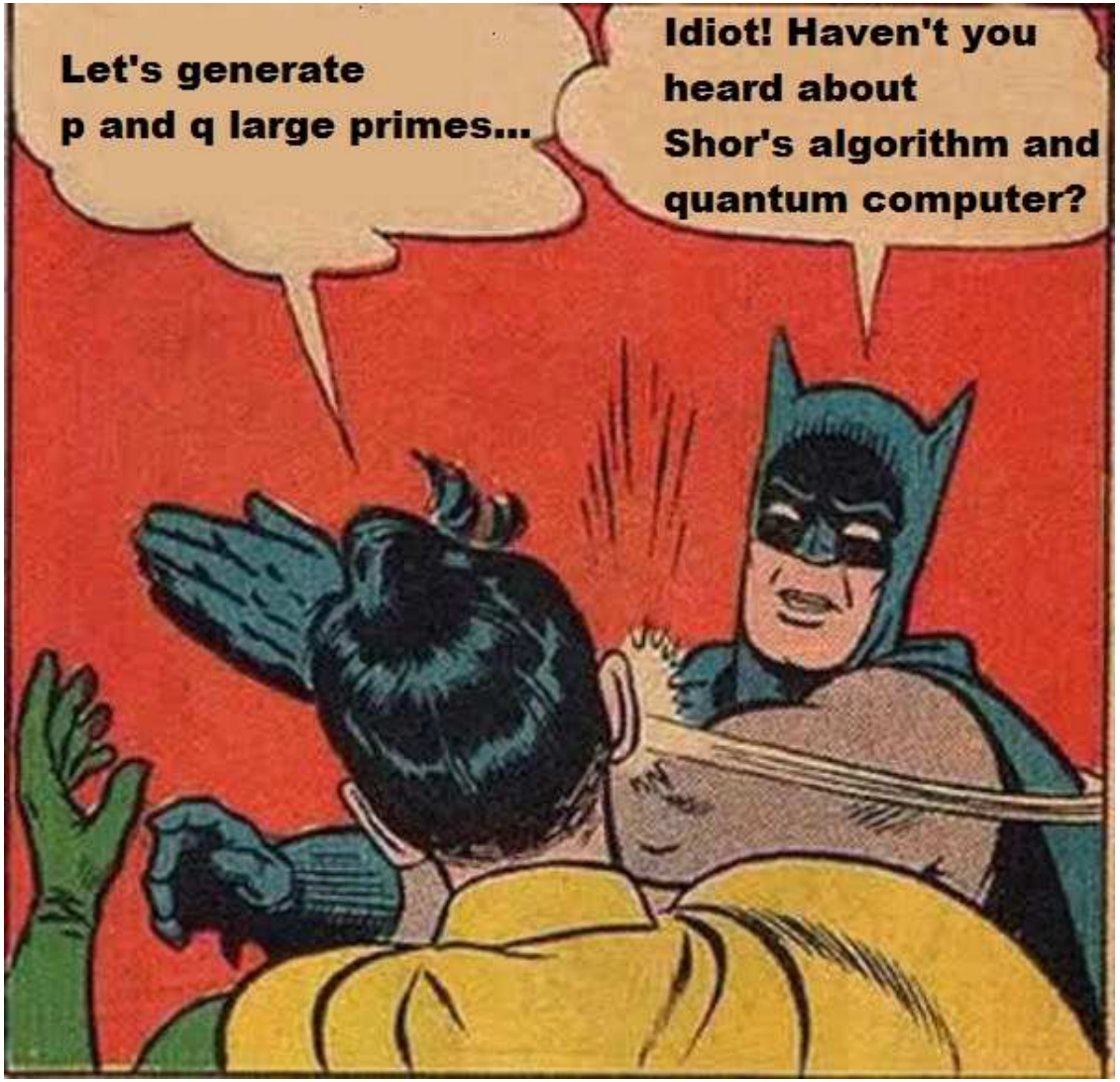
- collisions for 80-step (full) SHA-1 (shattered.io)
- using SHA-2 is recommended
- NIST selected Keccak (2012-10-02) as the basis for SHA-3

Administrator: Command Prompt

```
C:\ARON\HACK\collision\SHA-1>openssl dgst -sha256 -hex shattered-1.pdf
SHA256(shattered-1.pdf)= 2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha256 -hex shattered-2.pdf
SHA256(shattered-2.pdf)= d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-1.pdf
SHA1(shattered-1.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-2.pdf
SHA1(shattered-2.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-1_padding.pdf
SHA1(shattered-1_padding.pdf)= 3d24873f5e63555a41a764a866d14e903539688a

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-2_padding.pdf
SHA1(shattered-2_padding.pdf)= 3d24873f5e63555a41a764a866d14e903539688a

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-1_padding_Aron.pdf
SHA1(shattered-1_padding_Aron.pdf)= 796d9f4349c8f5acedf9d6918d3044b845db4817

C:\ARON\HACK\collision\SHA-1>openssl dgst -sha1 -hex shattered-2_padding_Aron.pdf
SHA1(shattered-2_padding_Aron.pdf)= 796d9f4349c8f5acedf9d6918d3044b845db4817

C:\ARON\HACK\collision\SHA-1>_
```

Algorithm anatomy

pqcrypto



Algorithm anatomy

Post-Quantum Cryptography: classical computers and quantum computers

„Imagine that it's fifteen years from now. Somebody announces that he's built a large quantum computer. RSA is dead. DSA is dead. Elliptic curves, hyperelliptic curves, class groups, whatever, dead, dead, dead. So users are going to run around screaming and say 'Oh my God, what do we do?' Well, we still have secret-key cryptography, and we still have some public-key systems. There's hash trees. There's NTRU. There's McEliece. There's multivariate-quadratic systems."

http://pqcrypto.org/

OK, but why?
Shor's algorithm (finding order of a group) and Grover's algorithm runs faster...

So, now what?

signature:	we can wait until real quantum computer is created...
encryption:	now it is already too late!
	Quantum-Safe Perfect Forward Secrecy (QSPFS)

Algorithm anatomy

pqcrypto

```
Shor's algorithm

        n = 15                                                  // to-be-factorized integer
        a = 7                                                   // random number, where "a < n"

        Calculate "r" (period/order):

                f(x) = a^x mod n = f(x + r) = a^(x + r) mod n
                1. 7^1 mod 15 = 7
                2. 7^2 mod 15 = 4
                3. 7^3 mod 15 = 13
                4. 7^4 mod 15 = 1
                5. 7^5 mod 15 = 7                               // f(1) = f(5)
                6. 7^6 mod 15 = 4
                7. ...

        r = 4                                                   // period/order
```

Algorithm anatomy

pqcrypto

```
Shor's algorithm

        Calculate "gcd(a^(r/2) +/- 1, n)" (greatest common divisor):

                p = gcd(48, 15) = ?
                q = gcd(50, 15) = ?

                //Euclidean algorithm:
                gcd(48, 15) => 48 = q[0] * 15 + r[0]            // q[0] = 3, r[0] = 3
                gcd(15, 3)  => 15 = q[1] * 3  + r[1]            // q[1] = 5, r[1] = 0
                The final non-zero remainder is r[0] = 3

                gcd(50, 15) => 50 = q[0] * 15 + r[0]            // q[0] = 3, r[0] = 5
                gcd(15, 5)  => 15 = q[1] * 5  + r[1]            // q[1] = 3, r[1] = 0
                The final non-zero remainder is r[0] = 5

        p = gcd(48, 15) = 3                                     // prime factor of "n = 15"
        q = gcd(50, 15) = 5                                     // prime factor of "n = 15"
```

Algorithm anatomy

ETSI: Quantum Safe Cryptography v1.0.0 (2014-10)

„[...] symmetric key algorithms like AES that can be broken faster by a quantum computer running Grover's algorithm than by a classical computer. [...] This is to say that AES-128 is as difficult for a classical computer to break as AES-256 would be for a quantum computer."

Table 1 - Comparison of conventional and quantum security levels of some popular ciphers.

Algorithm	Key Length	Effective Key Strength / Security Level	
		Conventional Computing	Quantum Computing
RSA-1024	1024 bits	80 bits	0 bits
RSA-2048	2048 bits	112 bits	0 bits
ECC-256	256 bits	128 bits	0 bits
ECC-384	384 bits	256 bits	0 bits
AES-128	128 bits	128 bits	64 bits
AES-256	256 bits	256 bits	128 bits

Note : Effective key strength for conventional computing derived from NIST SP 800-57 "Recommendation for Key Management"

Algorithm anatomy

E-GROUP
SOFTWARE & BEYOND

Post-Quantum Cryptography: classical computers and quantum computers

Hash-based:
- hash functions, HMAC structures are quantum safe
- signature (Lamport, Merkle etc.)

Lattice-based:
- shortest/closest vector problem (László Lovász, Miklós Ajtai)
- signature, encryption (GGH, NTRU etc.)

Multivariate equations-based:
- signature (UOV, Oil and Vinegar etc.)

Code-based:
- syndrome decoding problem, error-correcting codes
- signature, encryption (McEliece, Niederreiter etc.)

Symmetric key-based:
- encryption (AES, Twofish etc.)

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

„Websites can use TLS to secure all communications between their servers and web browsers." (... and other client softwares)

„It builds on the earlier SSL specifications (1994, 1995, 1996) developed by Netscape Communications for adding the HTTPS protocol to their Navigator web browser."

version	published in	published by
SSLv1.0	unpublished	Netscape
SSLv2.0	1995	Netscape
SSLv3.0	1996	Netscape
TLSv1.0	1999	IETF RFC 2246
TLSv1.1	2006	IETF RFC 4346
TLSv1.2	2008	IETF RFC 5246
TLSv1.3	2018	IETF RFC 8446

https://en.wikipedia.org/wiki/Transport_Layer_Security

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

OpenSSL

- cryptographic primitives
- CA functions
- SSL/TLS protocol

- multiple supported platforms

- command line interface (CLI)

- open source

- FIPS 140-2 level 1

OpenSSL™
Cryptography and SSL/TLS Toolkit

OpenSSL FIPS Object Module SE
Version 2.0.16
By
OpenSSL Validation Services

OpenSSL FIPS 140-2 Security Policy
Version 2.0.16

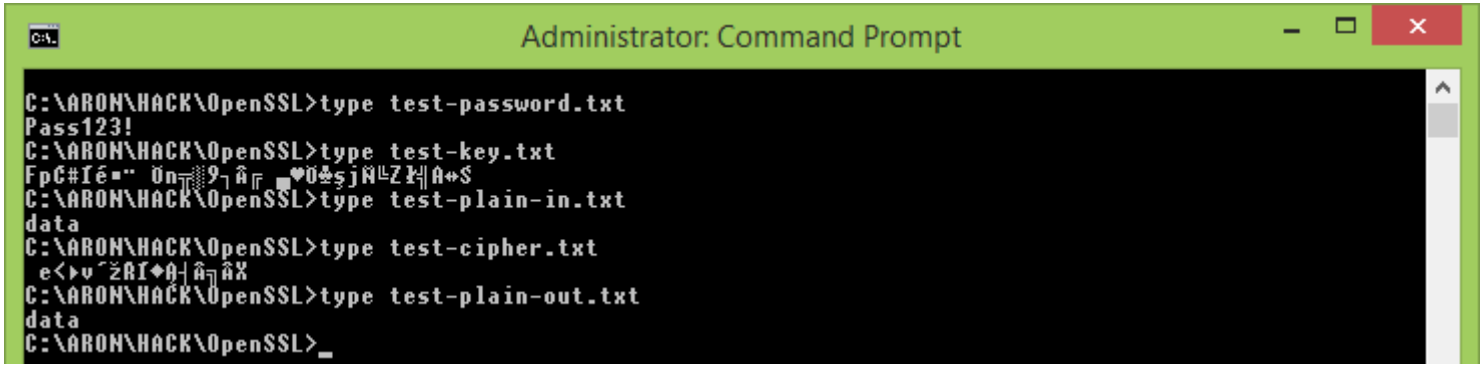
December 26, 2017

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
openssl dgst
        -sha256 -binary -out test-key.txt test-password.txt
openssl enc
        -aes-256-cbc
        -e -nosalt -kfile test-key.txt -in test-plain-in.txt -out test-cipher.txt

openssl dgst
        -sha256 -binary -out test-key.txt test-password.txt
openssl enc
        -aes-256-cbc
        -d -nosalt -kfile test-key.txt -in test-cipher.txt -out test-plain-out.txt
```

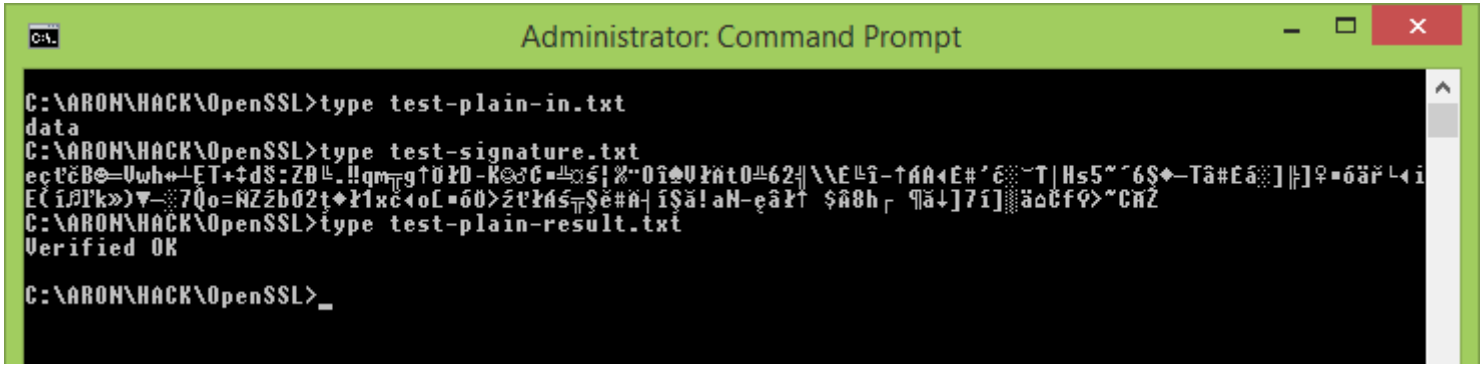


Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
openssl pkcs12
        -in test-user.p12 -out test-user.pem -passin pass:1234 -passout pass:1234
openssl rsa
        -in test-user.pem -passin pass:1234 -out test-user-private-plain.key
openssl rsa
        -in test-user.pem -passin pass:1234 -out test-user-public-plain.key -pubout
openssl dgst
        -sha256 -sign test-user-private-plain.key
        -out test-signature.txt test-plain-in.txt
openssl dgst
        -sha256 -verify test-user-public-plain.key -signature test-signature.txt
        -out test-plain-result.txt test-plain-in.txt
```



Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

Apache HTTP Server

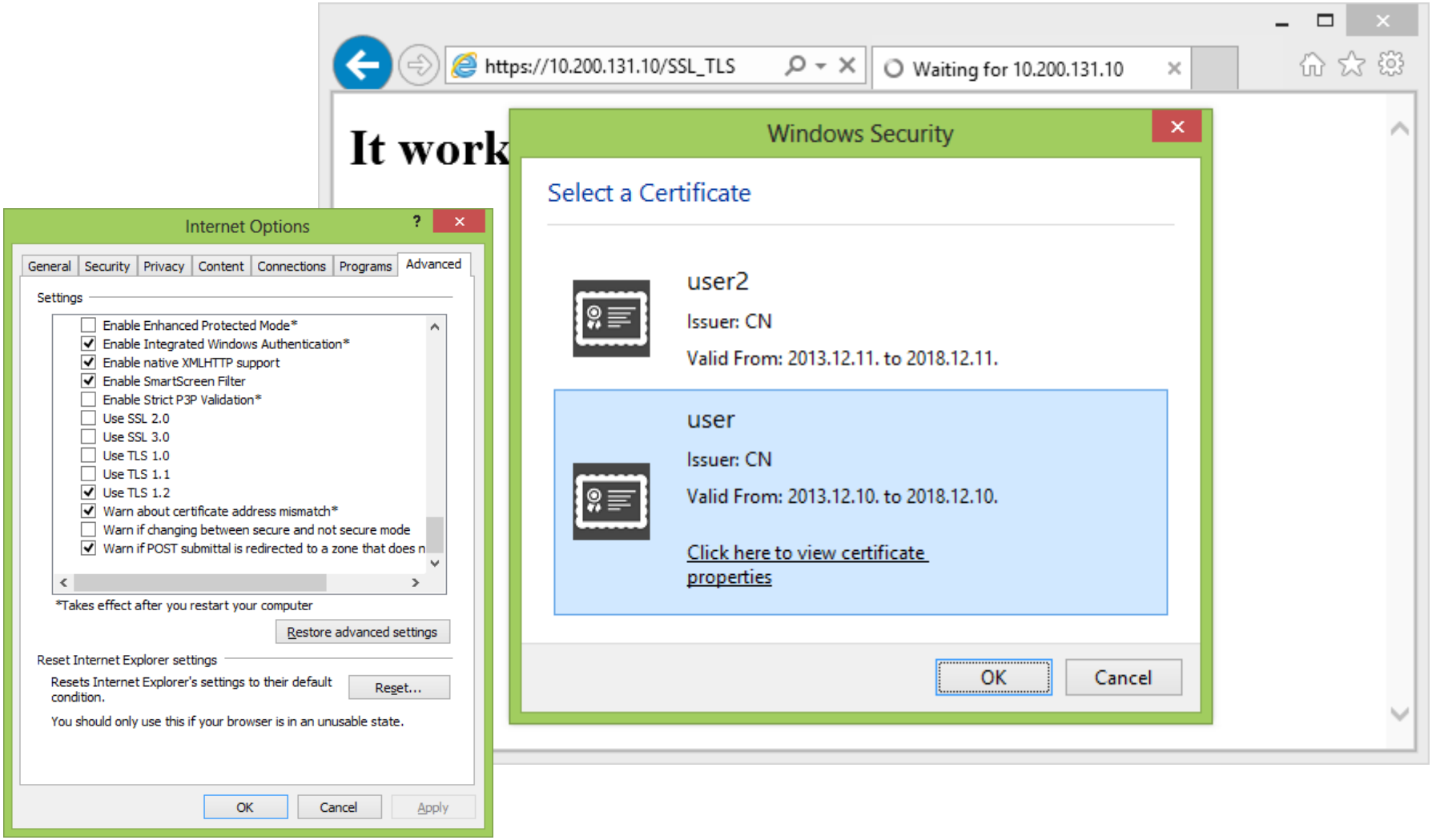
SSLEngine	on (SSL/TLS enabled)
SSLCipherSuite	HIGH:!DSS:!3DES (set of permitted algorithms)
SSLProtocol	+TLSv1.2 (set of permitted protocol versions)
SSLCompression	off (compression of SSL/TLS layer)
SSLInsecureRenegotiation	off (proper management of ClientHello and session)
SSLCertificateFile	my-server.crt (SSL/TLS server certificate)
SSLCertificateKeyFile	my-server.key (SSL/TLS server private key)
SSLCACertificateFile	my-server-ca.crt (CA certificate „whitelist”)
SSLOptions	+StdEnvVars +ExportCertData (handshake parameters)
SSLVerifyClient	require („client authentication”)
SSLVerifyDepth	1 (CA certificate chain length)

LoadModule	ssl_module	modules/mod_ssl.so
#LoadModule	deflate_module	modules/mod_deflate.so

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS



Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
openssl s_client
        -connect 127.0.0.1:443
        -CAfile my-server-ca.crt
        -cert my-client.crt
        -key my-client.key
        -tls1_2
```

```
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 4096 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 31B3BF330ABFC87A1AAFBBC05F2564372AF5716D32B0F0CCBC3BEFBE3FA2EB4E

    Session-ID-ctx:
    Master-Key: DED7E14BF7766F4DC3B2BF925435611CB4E2C8A01134D7C31865F35FD7EB0B78
BFD6FAA76B2EB7BD811F688E21A79B8E
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - bf df 54 7e ef b2 6d 84-40 ba f1 49 a1 da 16 46   ..T~..m.@..I...F

    0720 - 5a 44 46 a1 3f 12 9a ca-e3 84 77 7c ee 80 31 4b   ŹDF.?.....w¦..1K

    Start Time: 1386843096
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
openssl s_client
        -connect 127.0.0.1:443
        -CAfile my-server-ca.crt
        -cert my-client.crt
        -key my-client.key
        -tls1_2
```

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 132
  ⊟ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 128
      Version: TLS 1.2 (0x0303)
    ⊞ Random
      Session ID Length: 0
      Cipher Suites Length: 42
    ⊞ Cipher Suites (21 suites)
      Compression Methods Length: 1
    ⊞ Compression Methods (1 method)
      Extensions Length: 45
    ⊞ Extension: renegotiation_info
    ⊞ Extension: elliptic_curves
    ⊞ Extension: ec_point_formats
    ⊞ Extension: signature_algorithms
    ⊞ Extension: SessionTicket TLS
```

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ClientHello;
```

```
Client                                          Server

ClientHello          -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                     <--------         ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished             -------->
                                    [ChangeCipherSpec]
                     <--------             Finished
Application Data     <------->     Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 53
  ⊟ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 49
      Version: TLS 1.2 (0x0303)
    ⊞ Random
      Session ID Length: 0
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
      Compression Method: null (0)
      Extensions Length: 9
    ⊞ Extension: renegotiation_info
    ⊞ Extension: SessionTicket TLS
```

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ServerHello;
```

```
Client                                          Server

ClientHello        -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                   <--------          ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished           -------->
                                    [ChangeCipherSpec]
                   <--------                 Finished
Application Data   <------->         Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3292
  ⊟ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3288
      Certificates Length: 3285
    ⊟ Certificates (3285 bytes)
        Certificate Length: 1654
      ⊞ Certificate (pkcs-9-at-emailAddress=info@cn.hu,id-at-commonName=10.200.131.10
        Certificate Length: 1625
      ⊞ Certificate (pkcs-9-at-emailAddress=info@cn.hu,id-at-commonName=CN,id-at-orga
```

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```

```
Client                                          Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 514
  ⊟ RSA Encrypted PreMaster Secret
      Encrypted PreMaster length: 512
      Encrypted PreMaster: b25f8962c048dd691eba08d0d14ed747b67b5211cb859de4...
```

```
struct {
    select (KeyExchangeAlgorithm) {
        case dh_anon:
            ServerDHParams params;
        case dhe_dss:
        case dhe_rsa:
            ServerDHParams params;
            digitally-signed struct {
                opaque client_random[32];
                opaque server_random[32];
                ServerDHParams params;
            } signed_params;
        case rsa:
        case dh_dss:
        case dh_rsa:
            struct {} ;
    };
} ServerKeyExchange;
```

```
Client                                          Server

ClientHello        -------->
                                          ServerHello
                                         Certificate*
                                   ServerKeyExchange*
                                  CertificateRequest*
                   <--------         ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished           -------->
                                   [ChangeCipherSpec]
                   <--------                 Finished
Application Data   <------->       Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Certificate Request
    Handshake Type: Certificate Request (13)
    Length: 122
    Certificate types count: 3
  ⊞ Certificate types (3 types)
    Signature Hash Algorithms Length: 32
  ⊞ Signature Hash Algorithms (16 algorithms)
    Distinguished Names Length: 82
  ⊞ Distinguished Names (82 bytes)
```

```
struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    SignatureAndHashAlgorithm
      supported_signature_algorithms<2^16-1>;
    DistinguishedName certificate_authorities<0..2^16-1>;
} CertificateRequest;
```

```
Client                                          Server

ClientHello         -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                    <--------          ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished            -------->
                                    [ChangeCipherSpec]
                    <--------                 Finished
Application Data    <------->         Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Server Hello Done
    Handshake Type: Server Hello Done (14)
    Length: 0
```

```
struct { } ServerHelloDone;
```

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 1651
    Certificates Length: 1648
  ⊟ Certificates (1648 bytes)
      Certificate Length: 1645
    ⊞ Certificate (pkcs-9-at-emailAddress=info@cn.hu,id-at-commonName=user
```

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```

```
Client                                          Server

ClientHello         -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                    <--------          ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished            -------->
                                    [ChangeCipherSpec]
                    <--------                 Finished
Application Data    <------->         Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 514
  ⊟ RSA Encrypted PreMaster Secret
      Encrypted PreMaster length: 512
      Encrypted PreMaster: b25f8962c048dd691eba08d0d14ed747b67b5211cb859de4...
```

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa:
            EncryptedPreMasterSecret;
        case dhe_dss:
        case dhe_rsa:
        case dh_dss:
        case dh_rsa:
        case dh_anon:
            ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

```
Client                                          Server

ClientHello        -------->
                                          ServerHello
                                         Certificate*
                                   ServerKeyExchange*
                                  CertificateRequest*
                   <--------         ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished           -------->
                                   [ChangeCipherSpec]
                   <--------                 Finished
Application Data   <------->       Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ Handshake Protocol: Certificate Verify
    Handshake Type: Certificate Verify (15)
    Length: 516
```

```
struct {
     digitally-signed struct {
         opaque handshake_messages[handshake_messages_length];
     }
} CertificateVerify;
```

```
Client                                          Server

ClientHello         -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                    <--------          ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished            -------->
                                    [ChangeCipherSpec]
                    <--------                 Finished
Application Data    <------->         Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

```
struct {
    enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

Protocol anatomy

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 80
    Handshake Protocol: Encrypted Handshake Message
```

```
struct {
    opaque verify_data[verify_data_length];
} Finished;

verify_data
   PRF(master_secret, finished_label, Hash(handshake_messages))
      [0..verify_data_length-1];
```

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

```
struct {
    enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

```
Client                                          Server

ClientHello         -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                    <--------          ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished            -------->
                                    [ChangeCipherSpec]
                    <--------                 Finished
Application Data    <------->         Application Data
```

Protocol anatomy

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 80
    Handshake Protocol: Encrypted Handshake Message
```

```
struct {
    opaque verify_data[verify_data_length];
} Finished;

verify_data
   PRF(master_secret, finished_label, Hash(handshake_messages))
      [0..verify_data_length-1];
```

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Protocol anatomy

SSL/TLS

```
⊟ TLSv1.2 Record Layer: Application Data Protocol: http
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 512
    Encrypted Application Data: e9d7751fd65eb832cd7ef00e72e425bed422334a587a3a86...
```

```
enum {
    change_cipher_spec(20), alert(21), handshake(22),
    application_data(23), (255)
} ContentType;

struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (SecurityParameters.cipher_type) {
        case stream: GenericStreamCipher;
        case block:  GenericBlockCipher;
        case aead:   GenericAEADCipher;
    } fragment;
} TLSCiphertext;

struct {
    opaque IV[SecurityParameters.record_iv_length];
    block-ciphered struct {
        opaque content[TLSCompressed.length];
        opaque MAC[SecurityParameters.mac_length];
        uint8 padding[GenericBlockCipher.padding_length];
        uint8 padding_length;
    };
} GenericBlockCipher;
```

```
Client                                          Server

ClientHello        -------->
                                           ServerHello
                                          Certificate*
                                    ServerKeyExchange*
                                   CertificateRequest*
                   <--------           ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished           -------->
                                    [ChangeCipherSpec]
                   <--------                 Finished
Application Data   <------->         Application Data
```

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

2009-11-04: renegotiation attack
- affected: SSLv3.0 – TLSv1.2
- bug: ClientHello termination is needed
- fixed by: IETF RFC 5746

2011-09-23: BEAST
- affected: SSLv3.0 – TLSv1.0
- bug: CBC vulnerability, padding oracle
- fixed by: TLSv1.1

2012-09-13: CRIME
- affected: SSLv3.0 – TLSv1.2
- bug: TLS-compression leaks information
- fixed by: disable TLS-compression

2013-02-04: Lucky 13
- affected: SSLv3.0 – TLSv1.2
- bug: CBC vulnerability, padding oracle (time-based)
- fixed by: modify SSLCipherSuite

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

2013-03-12: RC4
affected: SSLv3.0 – TLSv1.2
bug: same messages leak information about the key
fixed by: enable AES, disable RC4

2013-08-01: BREACH
affected: SSLv3.0 – TLSv1.2
bug: HTTP-compression leaks information
fixed by: disable HTTP-compression

2014-10-14: POODLE
affected: SSLv3.0
bug: CBC vulnerability, padding oracle
fixed by: TLSv1.0

2015-03-03: FREAK
affected: SSLv3.0 - TLSv1.2
bug: RSA parameter downgrading
fixed by: modify SSLCipherSuite, update browser

Protocol anatomy

E-GROUP
SOFTWARE & BEYOND

SSL/TLS

2015-05-20: weakDH, Logjam

affected:	SSLv3.0 - TLSv1.2
bug:	Diffie-Hellman parameter downgrading
fixed by:	modify SSLCipherSuite, update browser

2016-03-01: DROWN

affected:	SSLv2.0
bug:	padding oracle, only RSA KeyExchange
fixed by:	disable SSLv2.0, modify SSLCipherSuite

Protocol anatomy

applied block cipher modes of operation: AES-CBC vs. AES-CTR vs. AES-GCM

- KRACK — Breaking WPA2 by forcing nonce reuse (2017-10-15)
  decrypt and replay messages of WPA2
  AES-CCMP — Counter Mode with CBC-MAC Protocol
  AES-GCMP — Galois/Counter Mode Protocol

"When the victim reinstalls the key, associated parameters such as the incremental transmit packet number (i.e. nonce) and receive packet number (i.e. replay counter) are reset to their initial value. Essentially, to guarantee security, a key should only be installed and used once."

https://www.krackattacks.com

```
data_plaintext_1.txt
313233343536373839303132333435363132333435363738393031323334353 6

data_plaintext_2.txt
6162636465666768696a6b6c6d6e6f706162636465666768696a6b6c6d6e6f70

data_plaintext_1.txt xor data_plaintext_2.txt
505050505050505050505a5a5e5e5a5a46505050505050505050505a5a5e5e5a5a46

data_ciphertext_1.txt
bd10464ad31fa0703c40dc86bcc757f2c76ea3d954da36c8905af9403eca48b6

data_ciphertext_2.txt
ed40161a834ff0206c1a86d8e29d0db4973ef389048a6698c000a31e609012f0

data_ciphertext_1.txt xor data_ciphertext_2.txt
505050505050505050505a5a5e5e5a5a46505050505050505050505a5a5e5e5a5a46

data_ciphertext_1.txt xor data_ciphertext_2.txt xor data_plaintext_2.txt = data_plaintext_1.txt
313233343536373839303132333435363132333435363738393031323334353 6
```

Thank you!

E-GROUP
SOFTWARE & BEYOND

Áron Szabó

mailto: aron.szabo@egroup.hu