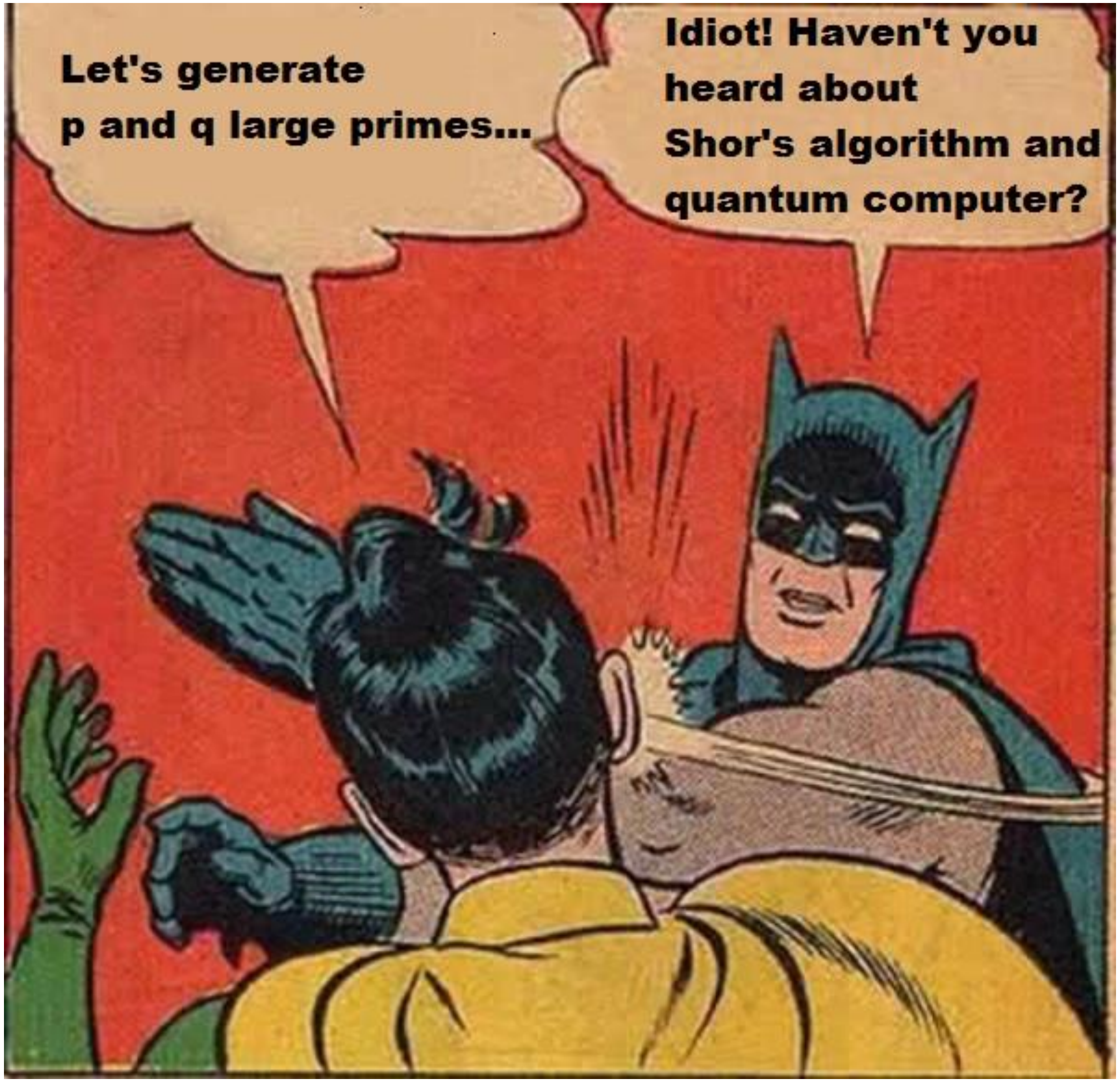
Post-Quantum Cryptography: prime questions = primary questions

HACKTIVITY

Áron SZABÓ

Hacktivity 2015
Budapest
October 9, 2015

About present and future

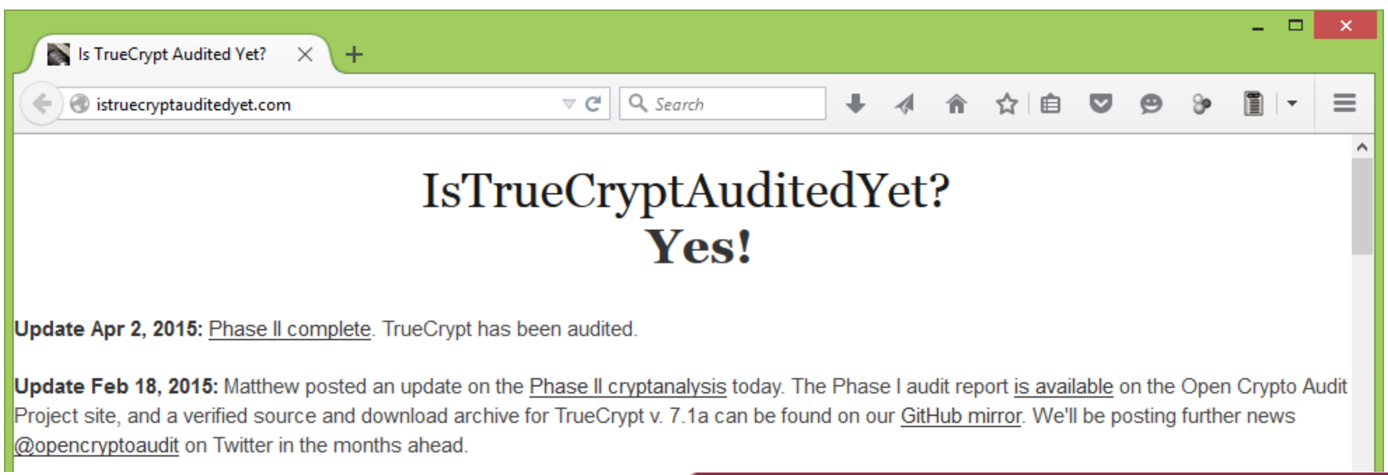


About present and future

Post-Quantum Cryptography: classical computers and quantum computers

mathematics of cryptographic primitives and algorithms
- MD5 collisions: Flame, modified signed Windows updates

implementations of cryptographic primitives and algorithms
- kleptography: Dual_EC_DRBG (PRNG)
- entropy: Android JCA (Bitcoin ECDSA), OpenSSL (PRNG)
- usage: ECDSA (Sony PlayStation 3)
  SSL/TLS: ECDHE vs. RSA (PRNG vs. PFS)
- code audits: TrueCrypt



About present and future

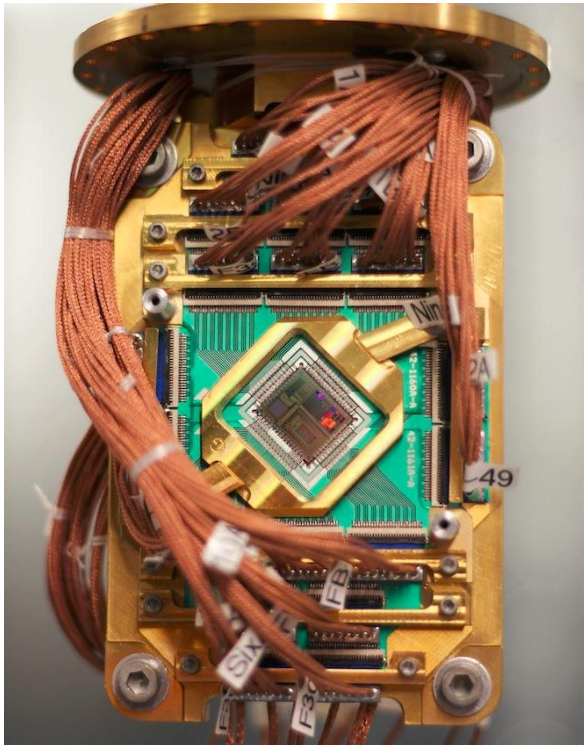
About present and future

About present and future

Post-Quantum Cryptography: classical computers and quantum computers

behaviours of cryptographic primitives and algorithms

- superposition: classical computer vs. quantum computers
- adiabatic qc: D-Wave Systems (sold to e.g. Lockheed Martin)



About security levels

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...
encryption: now it is already too late!
Quantum-Safe Perfect Forward Secrecy (QSPFS)

About security levels

```
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
```

About security levels

HACKTIVITY

```
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"
```

About security levels

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"

About standards and papers

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.)

About standards and papers

IETF: NTRU Cipher Suites for TLS (2001-07-03)
https://tools.ietf.org/html/draft-ietf-tls-ntru-00

IETF: Hash-Based Signatures (2014-07-04)
https://tools.ietf.org/html/draft-mcgrew-hash-sigs-02

ETSI: Quantum Safe Cryptography v1.0.0 (2014-10)
https://portal.etsi.org/Portals/0/TBpages/QSC/Docs/Quantum_Safe_White paper_1_0_0.pdf

IETF: XMSS: Extended Hash-Based Signatures (2015-03-23)
https://tools.ietf.org/html/draft-huelsing-cfrg-hash-sig-xmss-00

IETF: Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS) (2015-03-31)
https://tools.ietf.org/html/draft-housley-cms-mts-hash-sig-02

About standards and papers

Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louzia Papachristodoulou, Michael Schneider, Peter Schwabe, Zooko Wilcox-O'Hearn
SPHINCS: practical stateless hash-based signatures
http://sphincs.cr.yp.to/

Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, Tanja Lange, Mohamed Saied Emam Mohamed, Christian Rechberger, Peter Schwabe, Nicolas Sendrier, Frederik Vercauteren, Bo-Yin Yang
Post-Quantum Cryptography for Long-Term Security (2015-09-07)
http://pqcrypto.eu.org/docs/initial-recommendations.pdf

About hash-based algorithms

KISS („Keep It Simple Stupid”) principle has advantages:

Zooko Wilcox-O’Hearn: Tahoe-LAFS mailing list

„That is: a hash-based digital signature scheme can be broken if you can break the underlying secure hash function. All other digital signature schemes can be broken if you can break the secure hash function that they use for generating a message representative, *or* if you can break the digital signature scheme itself.”

Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen: Post-Quantum Cryptography

„To me hash-based cryptography is a convincing argument for the existence of secure post-quantum public-key signature systems.”

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- define the parameters of algorithm (e.g. LDWM_SHA256_M32_W1)

Parameter Name	Parameter Value
Name	LDWM_SHA256_M32_W1
H	SHA256 collision-resistant hash function H
F	SHA256 one way (preimage resistant) function F
m	32 the length in bytes of each element of an LDWM signature
n	32 the length in bytes of the result of the hash function
w	1 the Winternitz parameter; it is a member of the set { 1, 2, 4, 8 }
u	256 u = ceil(8 * n / w)
v	9 v = ceil((floor(lg((2^w - 1) * u)) + 1) / w)
ls	7 the number of left-shift bits used in the checksum function C ls = (number of bits in sum) - (v * w)
p	265 the number of m-byte string elements that make up the LDWM signature p = u + v

About hash-based algorithms

HACKTIVITY

LDWM (Lamport-Diffie-Winternitz-Merkle)

- generate private key chunks
- LDWM private key chunks can be applied to create signatures at most once!

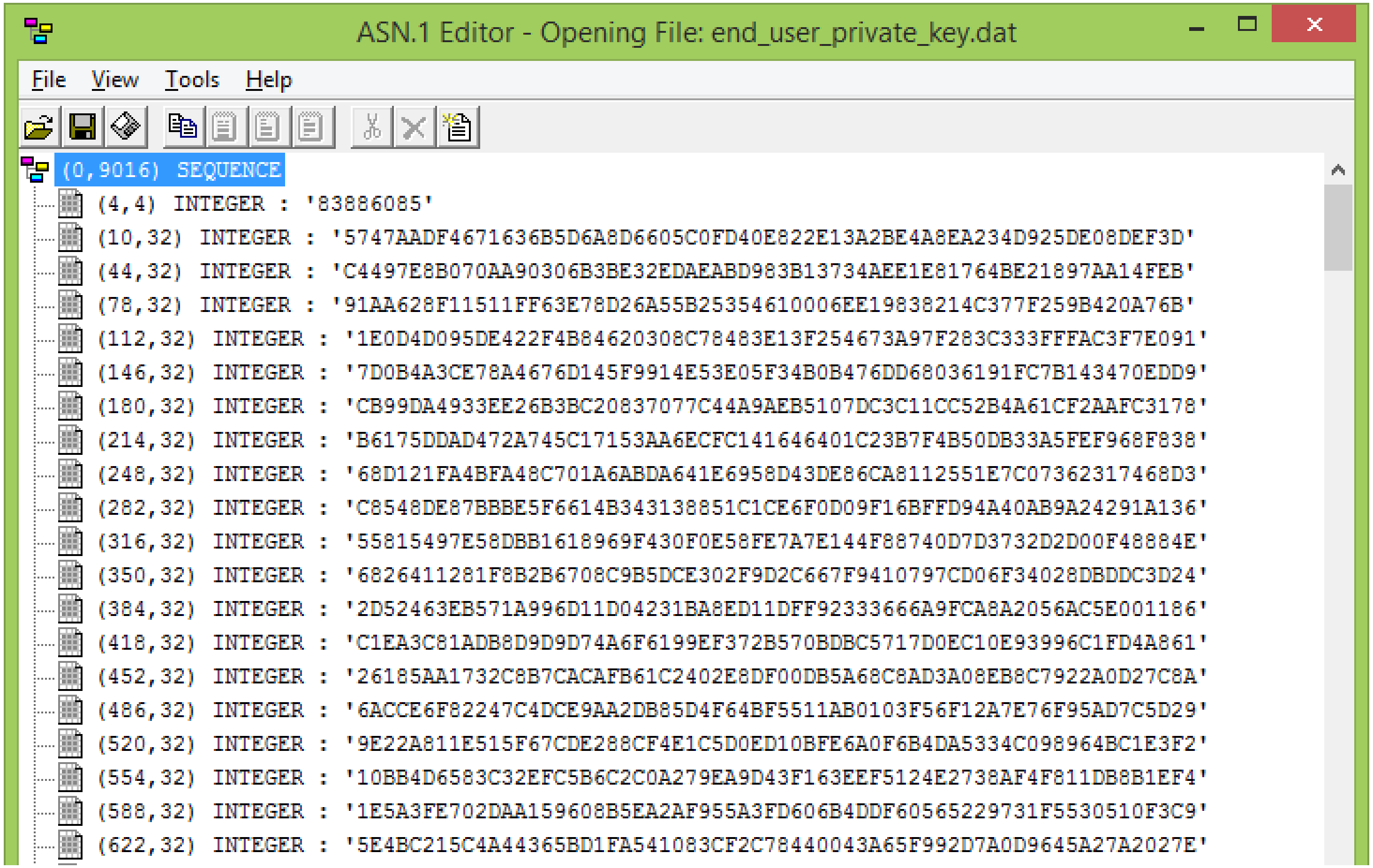
LDWM is a One-Time-Signature scheme (OTS)!

Key Element Index (i)	**OTS Private Key 0 Element (x[i]) calculated**
0	5747aadf4671636b5d6a8d6605c0fd40e822e13a2be4a8ea234d925de08def3d
1	c4497e8b070aa90306b3be32edaeabd983b13734aee1e81764be21897aa14feb
...	...
263	62f6f3e3a6fa657f8bf947ef236fc914faa47a3a141e330acc5d6d59d807e094
264	a4dc636971aefedee24820ed22f18849c8a28d28911d841c969af589de1d6397

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- key structure contains: ID of LDWM parameter set, private key chunks
  e.g. 83886085: ldwm_sha256_m32_w1 = 0x05000005 (IETF)



About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- derive public key chunks from private key chunks (hash function) concatenate public key chunks and hash them to get public key

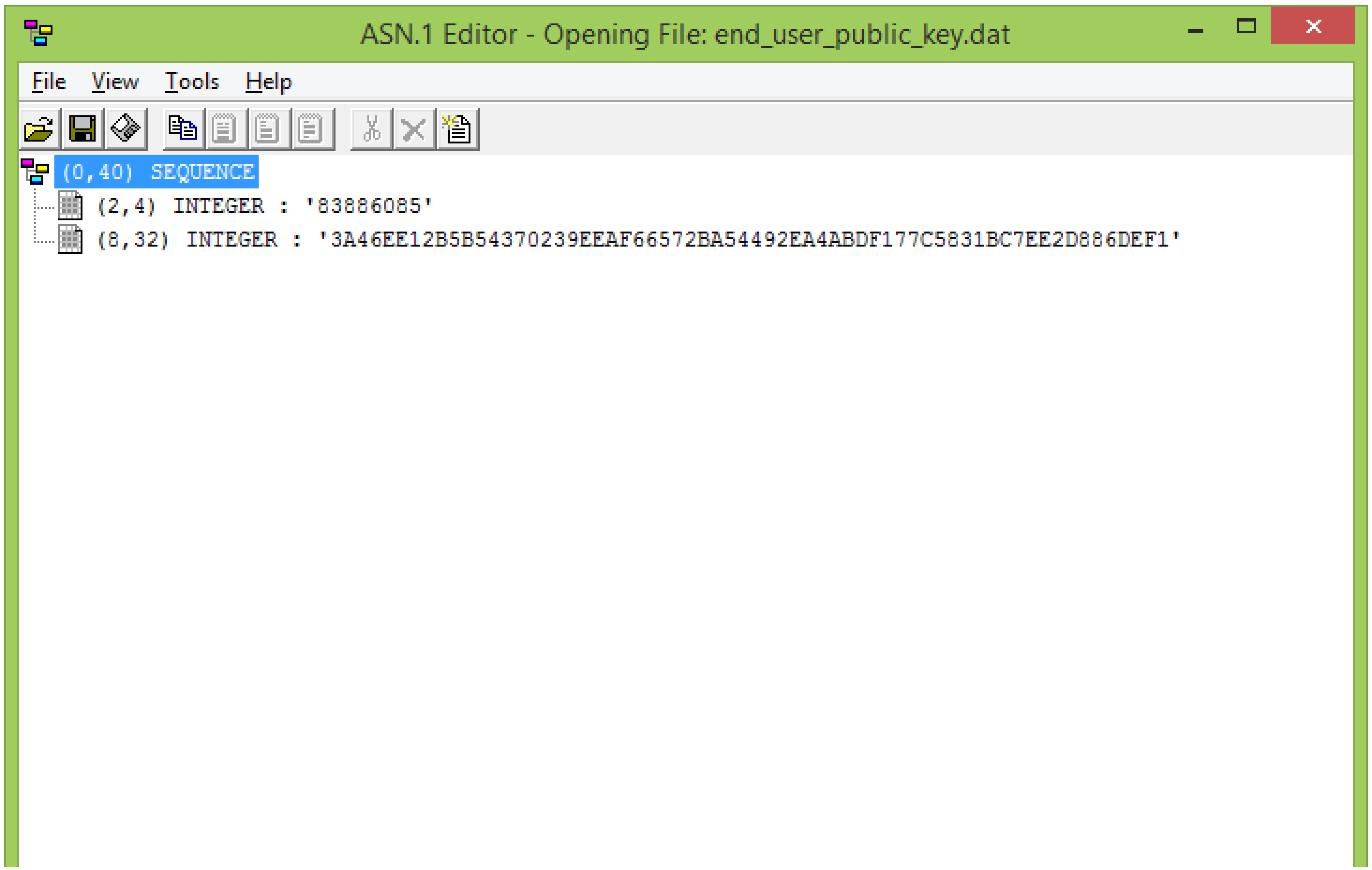
- Winternitz: y[i] = F^e(x[i]), where e = 2^w – 1
  „The parameter w can be chosen to set the number of bytes in the signature […] LDWM with w=1 has been shown to be existentially unforgeable under an adaptive chosen message attack […].”

<table>
<tr><th>Key Element Index
(i)</th><th>OTS Public Key 0 Element
(y[i] = F^1(x[i]))
calculated</th></tr>
<tr><td>0</td><td>f085b6d2ef122f1809b070ac52d49e8b1b16c2c8ef413daef2cb779a56ff9452</td></tr>
<tr><td>1</td><td>28e017a33bc500da868ed6c2116f25ed46328a69e0fc60588f5110355abaa191</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>263</td><td>ca6728f79d7ecd3a42ebcef1c2706e466b47daca09e49ddc16d2744468cce9b3</td></tr>
<tr><td>264</td><td>372045a7cce775ee6fb0b66f89c6b6aef6b93ece08a84f0125b5034f92b9bf0b</td></tr>
<tr><td></td><td></td></tr>
<tr><th>Key Element Index
(i)</th><th>OTS Public Key 0
(H(y[0] || y[1] || ... || y[p-1]))
calculated</th></tr>
<tr><td>0</td><td>3a46ee12b5b54370239eeaf66572ba54492ea4abdf177c5831bc7ee2d886def1</td></tr>
</table>

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

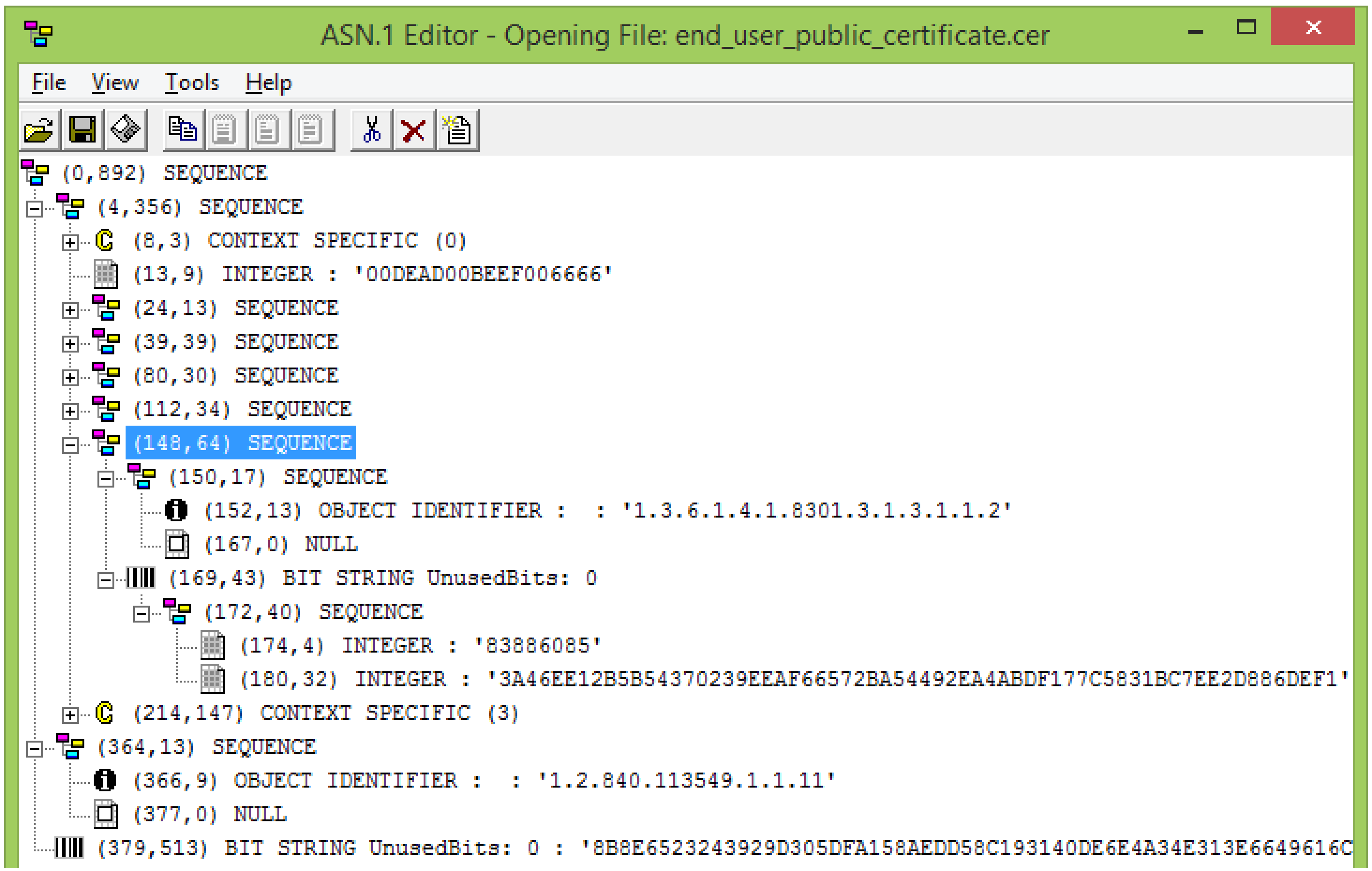
- key structure contains: ID of LDWM parameter set, public key
  e.g. 83886085: ldwm_sha256_m32_w1 = 0x05000005 (IETF)



About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

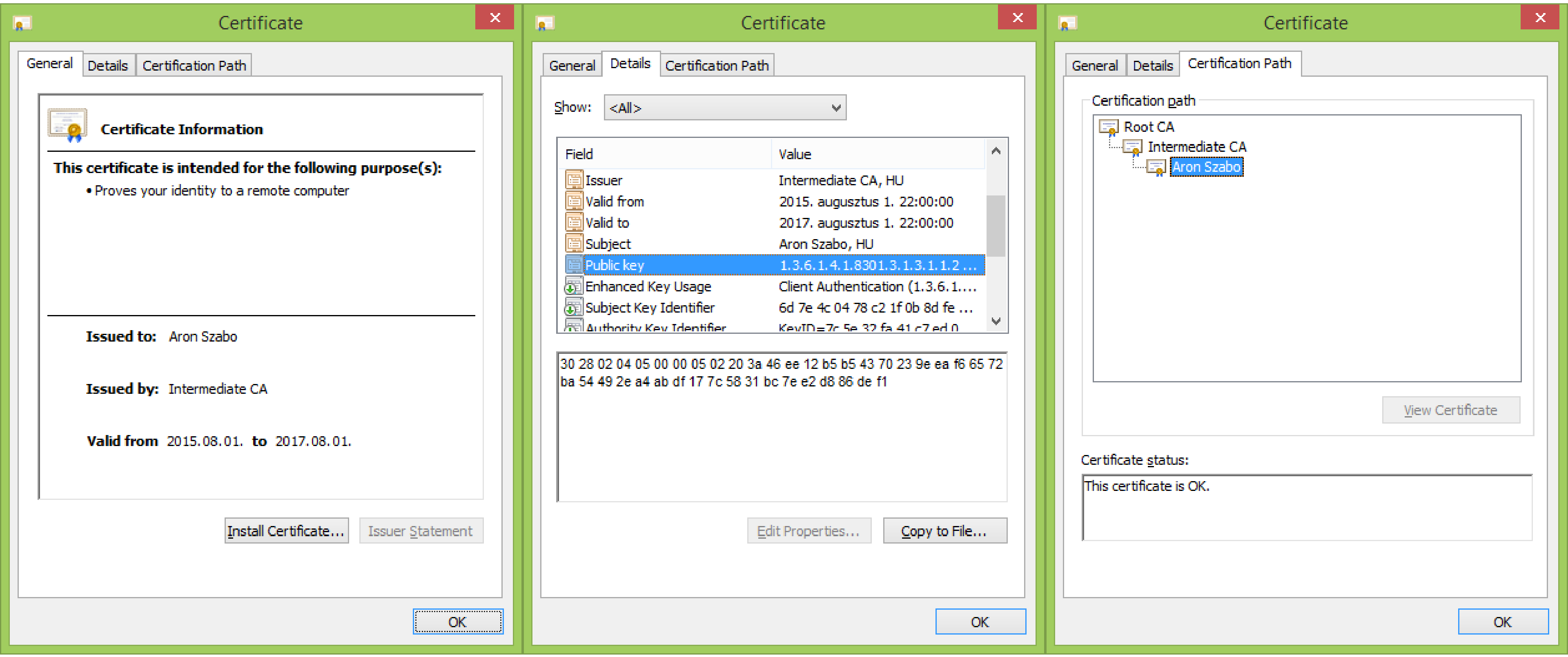
- X.509 certificate contains: ID of LDWM parameter set, public key, OID
  e.g. 1.3.6.1.4.1.8301.3.1.3.1.1.2: OTS with SHA-256 (TU Darmstadt)



About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- X.509 certificate contains: ID of LDWM parameter set, public key, OID
  e.g. 1.3.6.1.4.1.8301.3.1.3.1.1.2: OTS with SHA-256 (TU Darmstadt)



About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- the data to be signed is also needed, of course...
- this input data can be any binary string (as usual)...

Data Element Index (i)	ASCII Representation / Hexadecimal Byte Values
0	toBeSigned / 746f42655369676e6564
Data Element Index (i)	**Hexadecimal Byte Values**
0	24966de3536df15b186a13fe5ed8b8ad1def0439147d177a82ab88b65e91e4eb

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

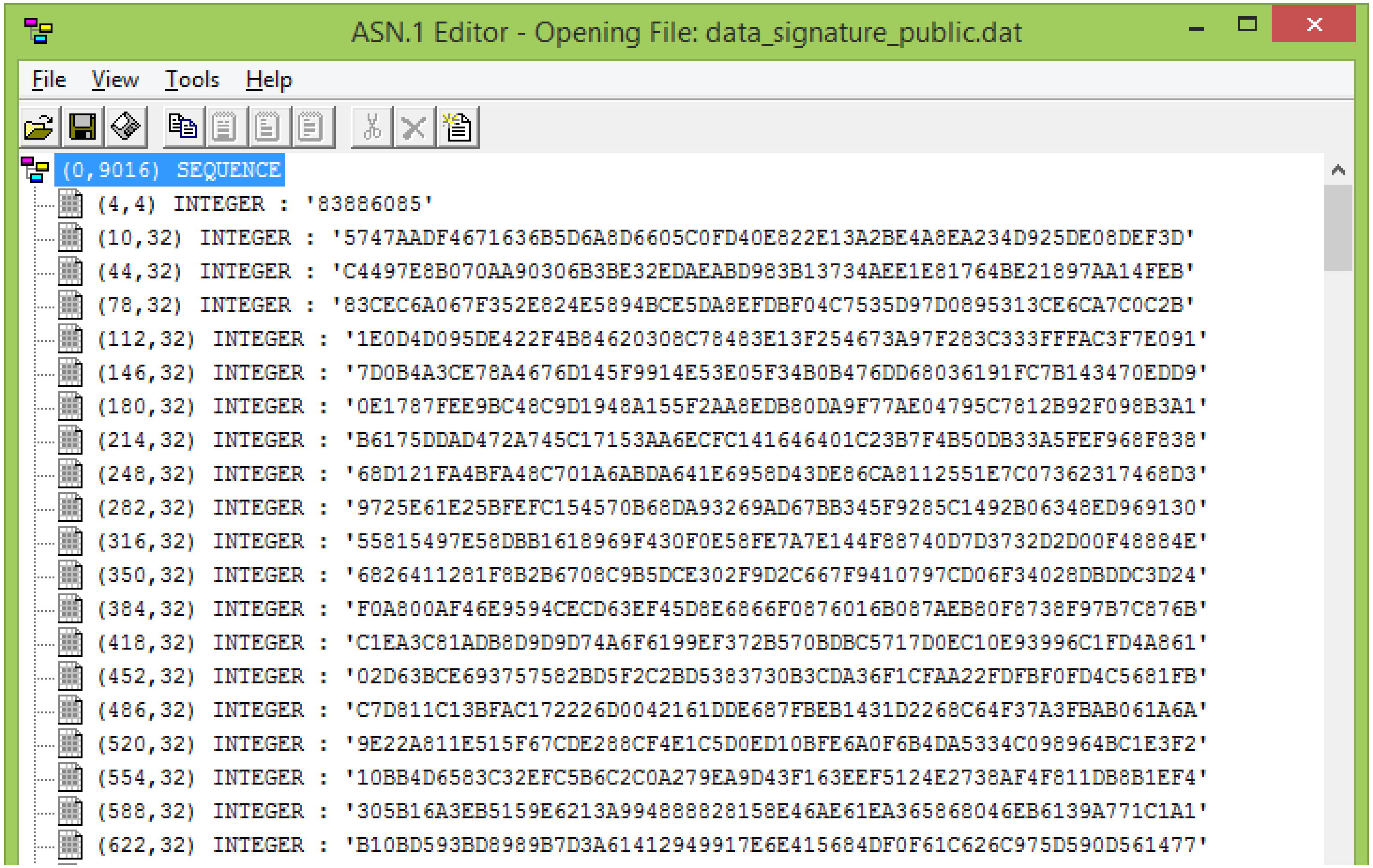
- derive signature chunks from private key chunks (hash function)
- checksum is also needed as input parameter to generate signature
- signature: y[i] = F^a(x[i]), where a = w-bit values based on H(message)

Data Element Index (i)	**OTS Checksum 0 (SUM((2^w - 1) - coef(S, i, w))) calculated**
0 (hex)	7a
0 (int)	122
OTS Element Index (i)	**OTS Signature 0 Element (y[i] = F^a(x[i])) calculated**
0	5747aadf4671636b5d6a8d6605c0fd40e822e13a2be4a8ea234d925de08def3d
1	c4497e8b070aa90306b3be32edaeabd983b13734aee1e81764be21897aa14feb
...	...
263	ca6728f79d7ecd3a42ebcef1c2706e466b47daca09e49ddc16d2744468cce9b3
264	a4dc636971aefedee24820ed22f18849c8a28d28911d841c969af589de1d6397

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- derive signature chunks from private key chunks (hash function)
  e.g. 83886085: ldwm_sha256_m32_w1 = 0x05000005 (IETF)



About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

- derive public key chunks from signature chunks (hash function)

```
if      (derived public key from signature ==
                retrieved public key from X.509 certificate)
{
        verification result is „successful”
}
else
{
        verification result is „failed”
}
```

OTS Element Index (i)	OTS Checksum 0 and OTS Signature 0 Verification Result calculated
0	TRUE

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

On one hand, be careful when...

- applying secure PRNG and secure hash function to generate key chunks
- managing private key states (deletion after signature creation: user/CA)

About hash-based algorithms

LDWM (Lamport-Diffie-Winternitz-Merkle)

... but on the other hand, it is

- ideal for human usage (such as on the field of e-government, e-voting)
- easy to implement and easy to use
- simple and secure, because cryptographic layer is based just on hash functions

Thank you!

HACKTIVITY

Áron SZABÓ

pqcrypto_LDWM.PHP
http://sourceforge.net/projects/pqcrypto-ldwm-php/