Digital Signatures: Mathematics

Zdeněk Říha

Data authentication

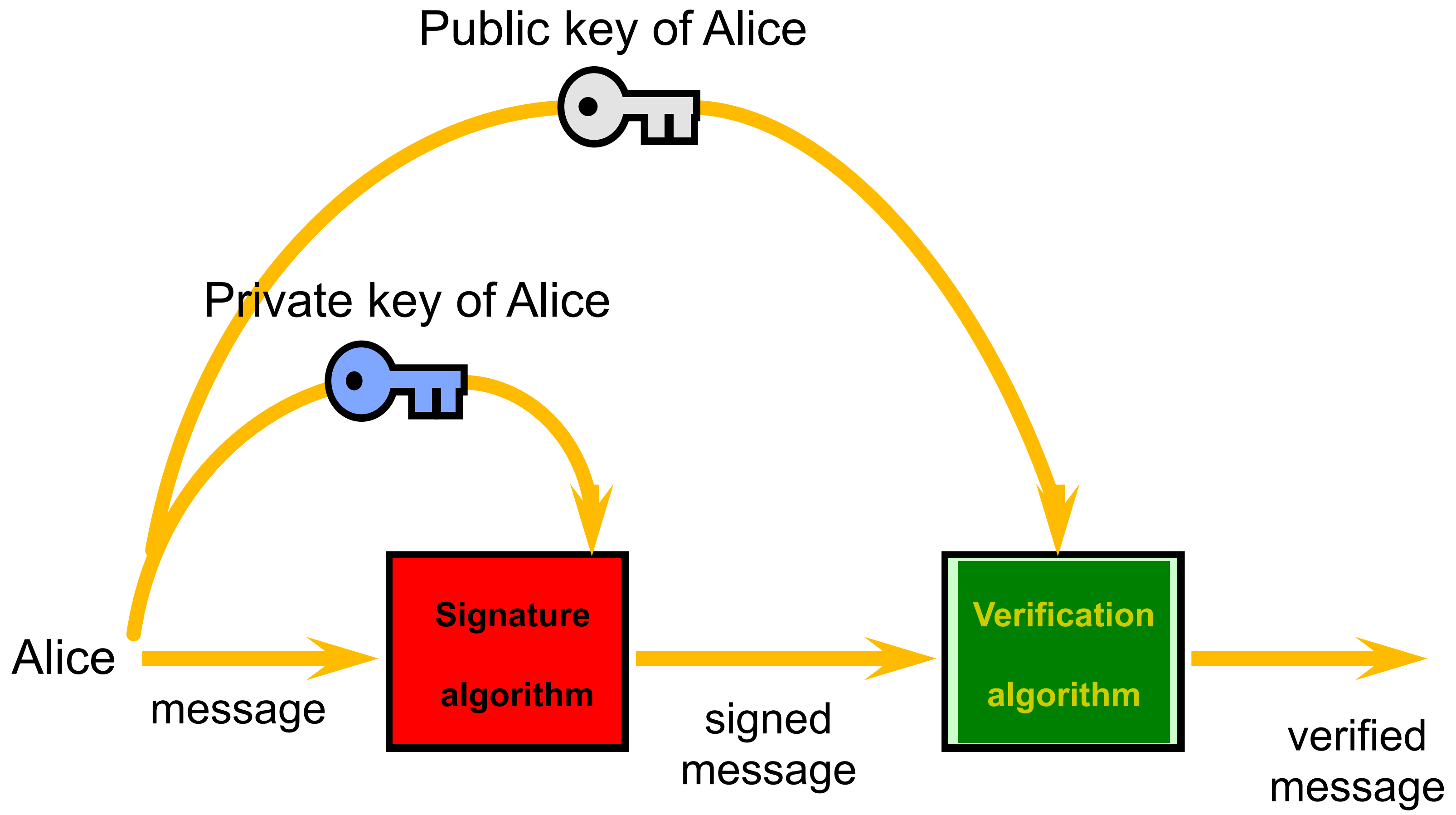
- Data integrity + data origin
- Digital signature
- Asymmetric cryptography
  - public and private key

Kerckhoffs' principle

- A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

- I.e. only the key should be kept secret, not the algorithm.

Digital signature scheme



Source: *Network and Internetwork Security* (Stallings)

Digital signature – keys

- Two keys per subject:
  - Private key for signature generation;
  - Public key for signature verification.
- Correct signatures can be generated only by those having the private key…
- To verify the signature we need the public key of the signed subject.
- The digital signature itself does not give any guarantees with respect to signing time.

Digital signature – algorithms

- First asymmetric cryptography algorithms appeared at the beginning of 1970s:
  - British GCHQ (Clifford Cocks).
  - Public announcement in 1997.
  - Application of the asymmetric algorithms for authentication - signature “invented” later by the academic community for their algorithms.
- First public algorithms at the end of 1970s (W. Diffie and M. Hellman influenced by R. Merkle).
- The famous algorithm RSA (Rivest, Shamir, Adelman) published in 1977, patented in 1983 (patent has already expired).
- Described in PKCS#1

Digital Signature Algorithm (DSA)

- Proposed in 1991 by NIST
- In 1994 the selection procedure for Digital Signature Standard (DSS) was concluded – DSA (Digital Signature Algorithm) was selected.
- Modified version of ElGamal algorithm, based on discrete logarithm in Z_p.
- Became FIPS standard FIPS 186 in 1993.
- Slightly modified in 1996 as FIPS 186-1.
- Extended in 2000 as FIPS 186-2.
- Updated in 2009 as FIPS 186-3 (new key sizes).

- Now NIST FIPS 186-3 supports RSA & DSA & ECDSA.

Elliptic curve DSA (ECDSA)

- Elliptic curves invented by Koblitz & Miller in 1985.
- ECDSA proposed in 1992 by Vanstone
- Became ISO standard (ISO 14888-3) in 1998
- Became ANSI standard (ANSI X9.62) in 1999

- ECDSA is a version of DSA based on elliptic curves.

RSA: mathematics

- Prime multiplication is simple & Factorization of integers is computationally intensive.
- We choose randomly 2 primes and compute n and φ(n) :
  - **p, q**
  - **$n = p \cdot q$**
  - **$\varphi(n) = (p-1)(q-1)$.**
- **e** is chosen such that **$\gcd(e, \varphi(n)) = 1$.**
- We compute **$d = e^{-1} \pmod{\varphi(n)}$.**
- Public key: **n, e.**
  Private parameters: **p, q, d.**
  Private key: **d.**
- Digital signature generation (decryption): s **$= m^d \bmod n$**
- Signature verification (encryption): **$m = s^e \bmod n$.**

- For RSA with 1024 bit **n**, the signature will be 1024 bit long.

RSA: example

- Intentionally small numbers (such cryptosystem is **not** secure).

- We generate parameters:
  - $p = 7927$, $q = 6997$,
  - $n = pq = 55465219$,
  - $\varphi(n) = 7926 \times 6996 = 55450296$.
- Public exponent is selected **$e = 5$**, equation is solved **$ed = 5d = 1 \pmod{55450296}$** to have **$d = 44360237$**.
- The public key: **($n = 55465219$, $e = 5$)**,
  The private key: **$d = 44360237$**.
- Digital signature generation:
  - Message $m =$ **31229978**
  - Signature $s = 31229978^{44360237} \bmod 55465219 = 30729435$.
- Signature verification:
  - Message should be $m = 30729435^{5} \bmod 55465219 =$ **31229978**.

RSA in practice: Padding

- $\mu(M)$ = 6b bb … bb ba || Hash(M) || 3x cc
  where x = 3 for SHA-1, 1 for RIPEMD-160
  - ANSI X9.31

- $\mu(M)$ = 00 01 ff … ff 00 || HashAlgID || Hash(M)
  - PKCS #1 v1.5

- $\mu(M)$ = 00 || H || $G(H) \oplus$ [*salt* || 00 … 00]
  where H = Hash(*salt*, M), *salt* is random, and G is a mask generation function
  - Probabilistic Signature Scheme (PSS)

RSA Padding example (PKCS#1 v1.5)

- Document
  - “00 01 02 03 04 05 06 07 07 06 05 04 03 02 01”
- Hash of the document (sha-1)
  - “b3 39 90 4c d2 a0 10 e6 19 37 eb e5 b5 83 37 8c 5d 10 51 95”
- Padded hash to be signed
  - “00 01 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 30 21 30 09 06 05 2b 0e 03 02 1a 05 00 04 14 b3 39 90 4c d2 a0 10 e6 19 37 eb e5 b5 83 37 8c 5d 10 51 95”

DSA: mathematics

- Key generation – domain parameters
  - Decide on a key length **L** and **N**, e.g. (1024,160).
    - N must be less than or equal to the hash output length
  - Choose an N-bit prime **q**. [“order of **g** w.r.t **p**”]
  - Choose an L-bit prime modulus **p** such that p–1 is a multiple of q.
  - Choose **g**, a number whose multiplicative order modulo p is q, e.g. $g = h^{(p-1)/q}$ mod *p* for some arbitrary *h* (1 < h < p-1). [“generator”]
  - Domain parameters **(p, q, g)** may be shared between different users of the DSA system.

DSA: mathematics II

- Key generation
  - Choose random **x**, such that $0 < x < q$.
  - Calculate **y** = $g^x \bmod p$.
- Private key: **x**.
- Public key: y & (p, q, g).

DSA: mathematics III

- Signature generation
  - Generate a random per-message value **k** such that $0 < k < q$.
  - Calculate **r** = $(g^{k} \bmod p) \bmod q$
  - Calculate **s** = $(k^{-1}(H(m) + x{*}r)) \bmod q$
  - The signature is (r, s).
- Signature verification
  - **w** = $(\mathbf{s})^{-1} \bmod q$
  - **u1** = $(\mathbf{H(m)}{*}w) \bmod q$
  - **u2** = $(\mathbf{r}{*}w) \bmod q$
  - **v** = $((g^{u1}{*}y^{u2}) \bmod p) \bmod q$
  - The signature is valid if **v = r**
- For DSA (1024,160) the signature size will be 2x160 bits.

DSA: Padding

- Decide on lengths **L** and **N**, e.g. (1024,160).
  - N must be less than or equal to the hash output length
    - E.g. for (1024,160) sha-1 is typically used,
      sha-256 would be ok as well and only first 160 bits would be used
  - s = (k^{-1}(**H(m)** + x*r)) mod q
- "It is recommended that the security strength of the (L, N) pair and the security strength of the hash function used for the generation of digital signatures be the same unless an agreement has been made between participating entities to use a stronger hash function. When the length of the output of the hash function is greater than N (i.e., the bit length of q), then the leftmost N bits of the hash function output block shall be used in any calculation using the hash function output during the generation or verification of a digital signature. A hash function that provides a lower security strength than the (L, N) pair ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function." [FIPS 186-3]

ECDSA: Elliptic curves

- Elliptic curve
- Points satisfying equation
  - $y^2 = x^3 + ax + b$
- And a point in infinity (∞)
- Set of points & operations form an Abelian group
- In ECC (elliptic curve cryptography) elliptic curves over finite fields are used
  - Prime fields: $\mathbf{F_p}$, **p** prime
  - Extension fields of characteristics 2: $\mathbf{F_{2^m}}$
- Finding the discrete logarithm of a random elliptic curve element with respect to a publicly-known base point is infeasible.

ECDSA: Elliptic curve domain parameters

- **(field,a,b,G,n,h)**
    - Finite field
        - **p** for F_p
        - **m, bases (trinomial, pentanomial)** for $\mathbf{F_{2^m}}$
    - Coefficients **a, b**: $y^2 = x^3 + ax + b$
    - Group generator: **G**
    - Order of the G: **n**
    - Optional cofactor: **h**
        - (h = number of elements in field / order n)
    - The base point G generates a cyclic subgroup of order n in the field.

ECDSA: Keys

- Generating key pair
  - Select a random integer **d** from [1,n − 1]
  - Compute **P** = d*G;
- Private key: **d**
- Public key: **P**

- For 256-bit curve
  - the private key **d** will be approx. 256-bit long
  - the public key **P** is a point on the curve – will be approx 512-bit long

ECDSA: Signatures

- Generate signature
  - Select a random integer **k** from [1,n − 1]
  - **(x_1,y_1)** = k*G
  - Calculate **r** = x_1 (mod n)
  - Calculate **s** = k^{-1}(M + r*d) (mod n)
  - Signature is **(r,s)**.
- Signature verification
  - Calculate **w** = s^{-1} (mod n)
  - Calculate **u_1** = z*w (mod n) & **u_2** = r*w (mod n)
  - Calculate **(x_1,y_1)** = u_1*G + u_2*P
  - The signature is valid if **r = x_1 (mod n)**.
- For 256-bit curve the signature length will be approx. 512 bits

ECDSA: Padding

- Rules are same as for DSA
- “It is recommended that the security strength associated with the bit length of n and the security strength of the hash function be the same unless an agreement has been made between participating entities to use a stronger hash function. When the length of the output of the hash function is greater than the bit length of n, then the leftmost n bits of the hash function output block shall be used in any calculation using the hash function output during the generation or verification of a digital signature. A hash function that provides a lower security strength than the security strength associated with the bit length of n ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function.” [FIPS 186-3]

Hash function

- **Cryptographic** hash function
- Input of arbitrary size
- Output of fixed size: n bits (e.g. 256 bits).
- Function is not injective (there are collisions).
- Hash is a compact representative of input (also called imprint, (digital) fingerprint or message digest).
- Hash functions often used to protect integrity. First the has is computed and then only the has is protected (e.g. digitally signed).

Hash function properties

- One-way property
  - It is easy to calculate **h(x)** for arbitraty **x**.
  - In a reasonable time it is not possible for the fixed **y** to find **x**, such that **h(x) = y**.
- Collision resistance
  - (weak): In a reasonable time it is not possible for a given **x** to find **x’** (**x ≠ x’**) such that **h(x) = h(x’)**.
  - (strong): In a reasonable time it is not possible to find any **x, x’** such that **h(x) = h(x’)**.

Cryptographic hash functions

- MD4: output 128 bits
  - not used anymore, serious weaknesses found.
- MD5: output 128 bits
  - Still used although not considered secure at all
  - Broken: efficient algorithm for finding collisions available
  - 128-bit output not considered secure enough
- SHA-1 (Secure Hash Algorithm): output 160 bits
  - NIST standard, used in DSS (Digital Signature Standard)
  - Not recommended for longer term use.

Cryptographic hash functions

- RIPEMD
  - Output : 128, 160, 256 or 320 bits
  - Less frequently used
- Whirlpool
  - Output: 512 bits
  - Based on AES
  - Recommended by NESSIE project
  - Standardized by ISO

Cryptographic hash functions

- **SHA-2**
  - SHA-256, SHA-384, SHA-512, SHA-224
  - Defined in FIPS 180-2
  - Recommended hash functions
- **SHA-3**
  - Selection of the future standard hash function is currently in progress
  - Winner expected in 2012

Hash functions - examples

- MD5
  - Input: „Autentizace“.
  - Output: 2445b187f4224583037888511d5411c7 .
  - Output 128 bits, written in hexadecimal notation.
  - Input: „Cutentizace“.
  - Output: cd99abbba3306584e90270bf015b36a7.
  - A single bit changed in input → big change in output, so called “Avalanche effect”
- SHA-1
  - Input: „Autentizace“.
  - Output: 647315cd2a6c953cf5c29d36e0ad14e395ed1776
- SHA-256
  - Input: „Autentizace“.
  - Output: a2eb4bc98a5f71a4db02ed4aed7f12c4ead1e7c98323fda8ecbb69282e4df584