PV181 Laboratory of security and applied cryptography

Asymmetric cryptography

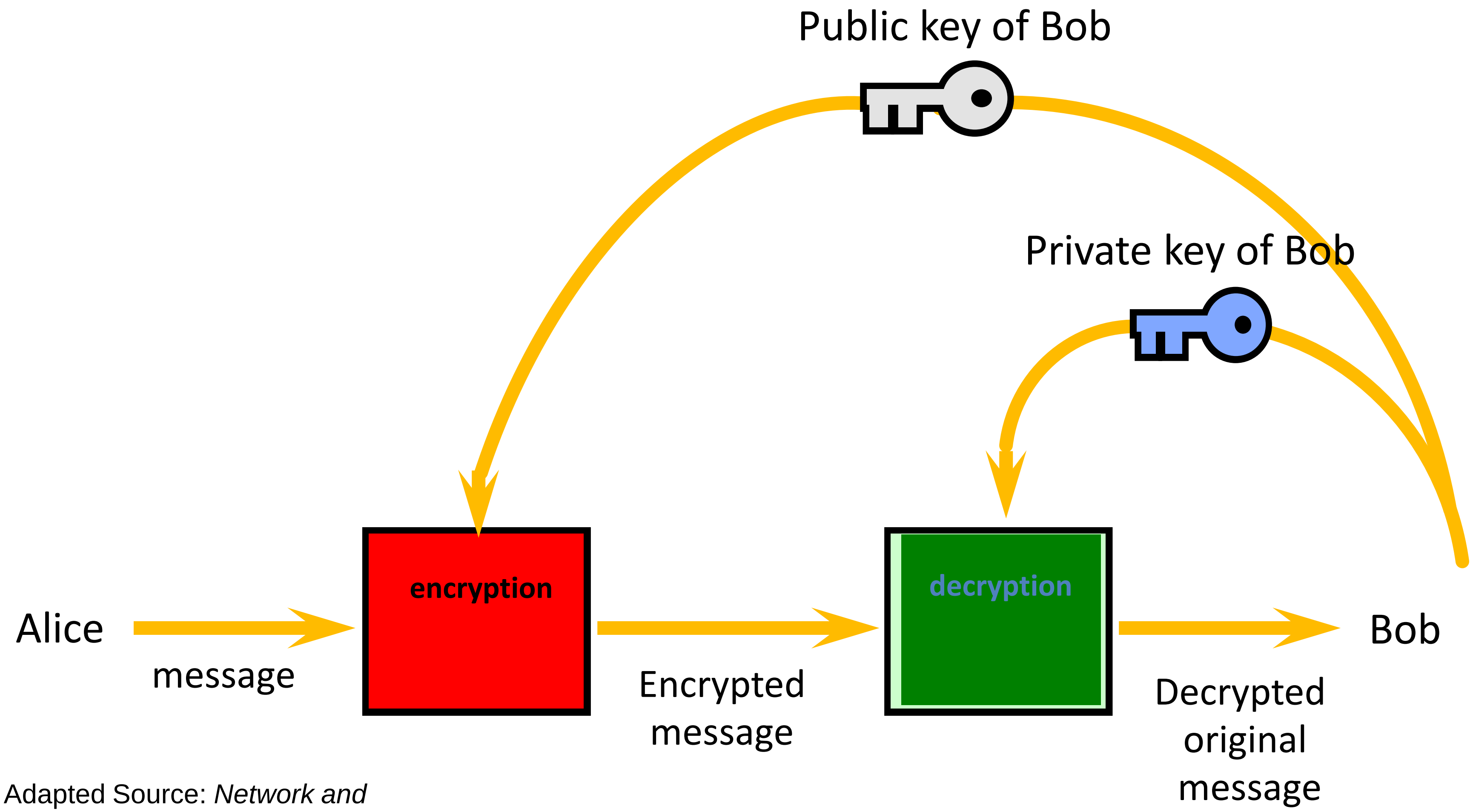
Marek Sýs, Zdeněk Říha

CRoCS
Centre for Research on Cryptography and Security

Asymmetric cryptography

- Confidentiality
  - **asymmetric cipher** - encryption, key agreement
- Authentication
  1. Entity – identity verification - **certificates**
  2. Data origin - **digital signature**
- Integrity
  - **digital signature**
- Non-repudiation
  - **digital signature**

Asymmetric cryptosystem



Adapted Source: *Network and Internetwork Security* (Stallings)

Asymmetric cryptography

- Two related keys – created by **one** party
  - different inverse operations (encryption - decryption, signing – signature verification)
- Properties - hard to compute private from public key
  - based on hard mathematical problems
- Hard problems and cryptosystems:
  - Integer factorization – RSA, Rabin, …
  - Discrete logarithm problem (DLP): ElGamal, EC, DSA, …
  - Others (DH, decoding,…) – Diffie-Helman, McElliece,…

Public vs private key cryptography

- Private (symmetric)
  - both parties share secret (**private**)
  - Pros: fast encryption
  - Cons: key distribution requires **secure channel**
- Public (asymmetric)
  - one key is **public**
  - Pros - key distribution – **insecure** channel is OK
  - Cons - slow encryption
- Practice - private + public:
  - **public** used to establish key for **private** key system

Diffie Hellman algorithm

- 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

Hard problems

- Integer factorization
  - for **n** find divisor **p** of **n**
- Discrete logarithm problem:
  - in $Z_{\boldsymbol{p}}$, Elliptic curves (EC)
    for $y = g * g * \cdots * g = g^x$ find x
  - $*$ represents operation (*, +) for given domain (integers, EC)
  - Domain parameters:
    - $g, n = ord(g)$ - n should be large
    - params defining algebraic structure: $Z_{\boldsymbol{p}}$ or **EC**

DLP for integers

- For g, p, y find integer x such that

$$y \equiv g^x \; mod \; p$$

$g = 2, p = 31$

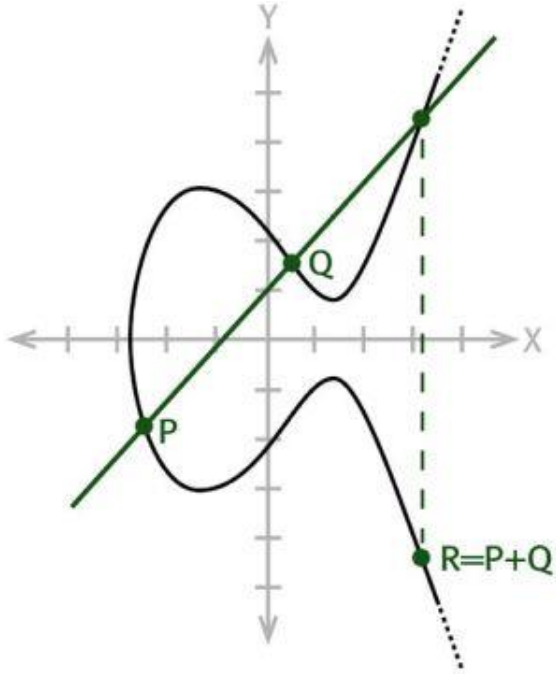
$g^x \; mod \; p$: $2,4,8,16,1 = 3^5 \; mod \; 31$ (order = 5)

$g = 3, p = 31$

$g^x \; mod \; p$: $3,9,27,19, \ldots, 3^{30} \; mod \; 31 = 1$

full order = 30, 3 is generator (all numbers)

Elliptic curve



-
- Inspired by EC in real numbers
  – operation point addition
- Defined by $\boldsymbol{a}$, $\boldsymbol{b}$ and field $\boldsymbol{Z_p}$ or $\boldsymbol{F_{2^m}}$
- Consist of 2D points $[x, y]$ + ∞ point:

  $y^2 = x^3 + ax + b$ holds in field
- NIST recommended 15 curves - P192, P224,…

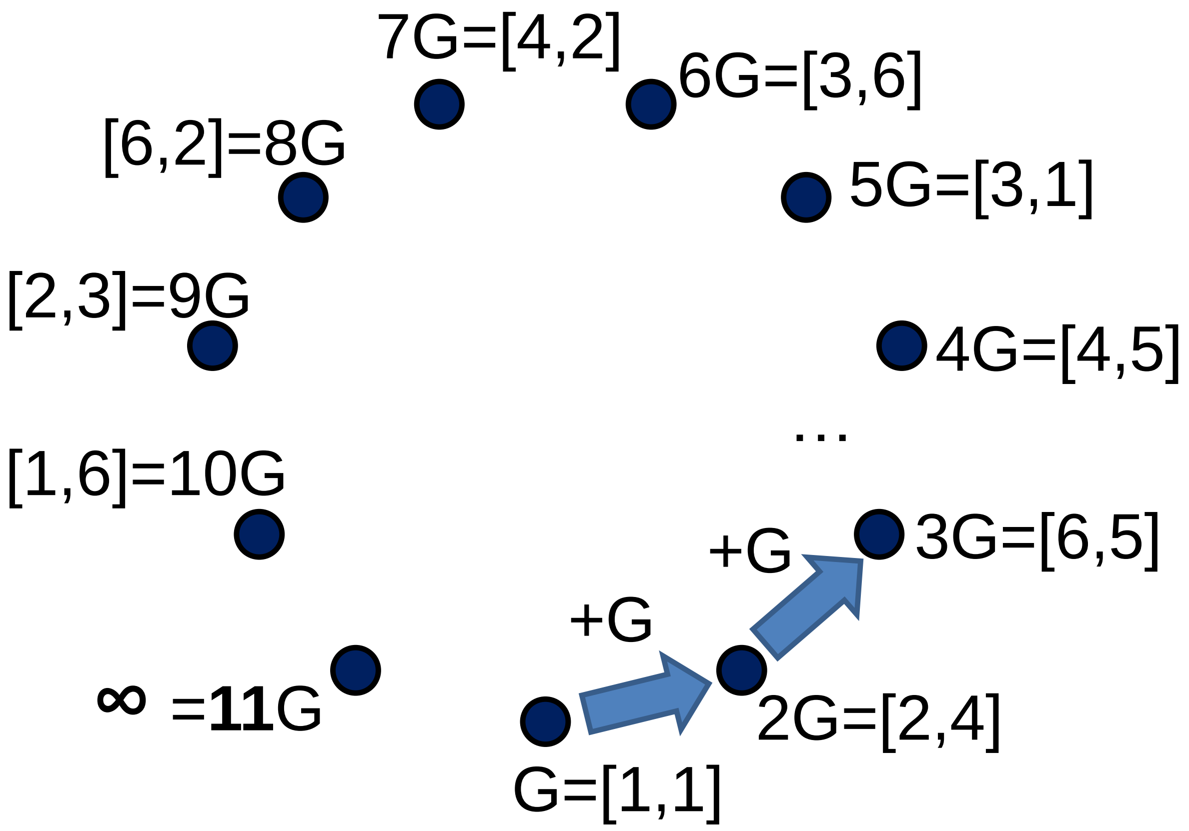
Example (domain params)

- $y^2 = x^3 + x + 6 \; mod \; 7$

Points:

[1, 1], [1, 6]
[2, 3], [2, 4]
[3, 1], [3, 6]
[4, 2], [4, 5]
[6, 2], [6, 5]
“Infinity”

G=[1,1]



RSA: mathematics

1. Two **random** large primes $\boldsymbol{p}$, $\boldsymbol{q}$
2. Public exponent e is chosen with:
$$\gcd(e, p-1) = \gcd(e, p-1) = 1$$
3. Private exponent d is computed:
$$e.d \equiv 1 \; mod \; (p-1)(q-1)$$

Encryption: $E(m) = m^e mod\; n \Rightarrow m'$ public key n, e

Decryption: $D(m') = m'^d mod\; n \Rightarrow m$ private key n, d

RSA-1024 means size of $\boldsymbol{n}$ = 1024 bits

- $m < n$, m' is typically size of $\boldsymbol{n}$

RSA example

- Intentionally small numbers (**not** secure).
- We generate parameters: p = 17, q = 7, $n = p.q$ = 119

- Public exponent is selected e **= 3,5** $(\gcd(3{,}7-1)=3)$
- Private exponent computed:
  ed = 5d = 1 (mod 96) to have **d = 77**.
- The public key: **(n = 119, e = 5)**,
  The private key: **(n = 119, d = 77)**

- Encryption/decryption:
  – Message **m** = ‘C’ = 65
  – Encryption **m’** = 65^5 mod 119 = 46.
  – Decryption m = 46^{77} mod 119 = 65

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)

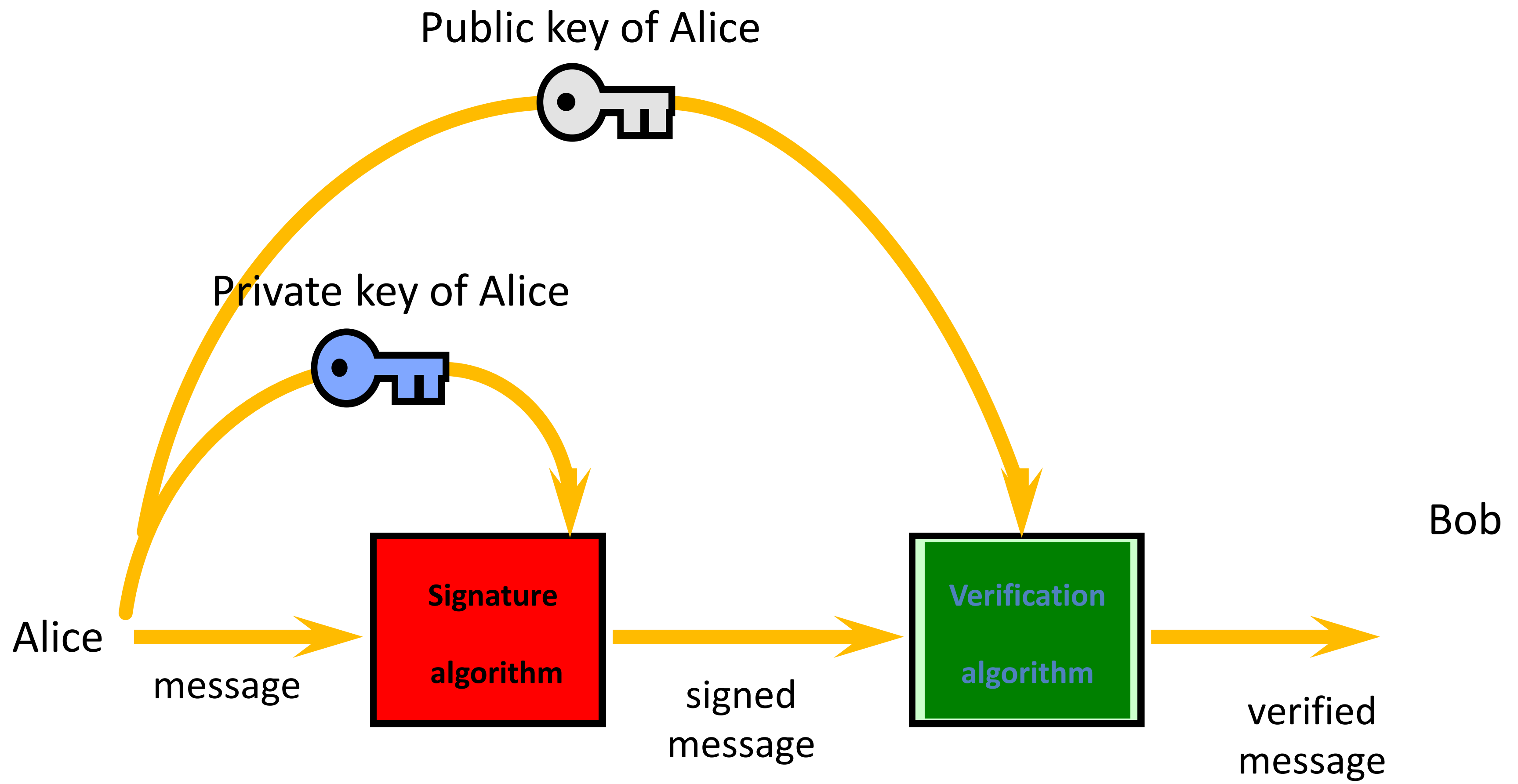
- Document
  - “00 01 02 03 04 05 06 07 07 06 05 04 03 02 01”
- Hash of the document (sha-1)
  - “b3 39 90 4c d2 a0 10 e6 19 37 eb e5 b5 83 37 8c 5d 10 51 95”
- Padded hash
  - “00 01 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 30 21 30 09 06 05 2b 0e 03 02 1a 05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

Digital signature

- Asymmetric cryptography
  - Private key – signature generation (usually only **hash** of data is signed **not** data itself)
  - Public key – verification procedure
- Data integrity + data origin + non-repudiation:
- Non-repudiation - correct signatures can be generated only by those having the private key

- The digital signature itself does not give any guarantees with respect to signing time.

Digital signature scheme



Source: *Network and Internetwork Security* (Stallings)

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.

Digital Signature Standard (DSS)

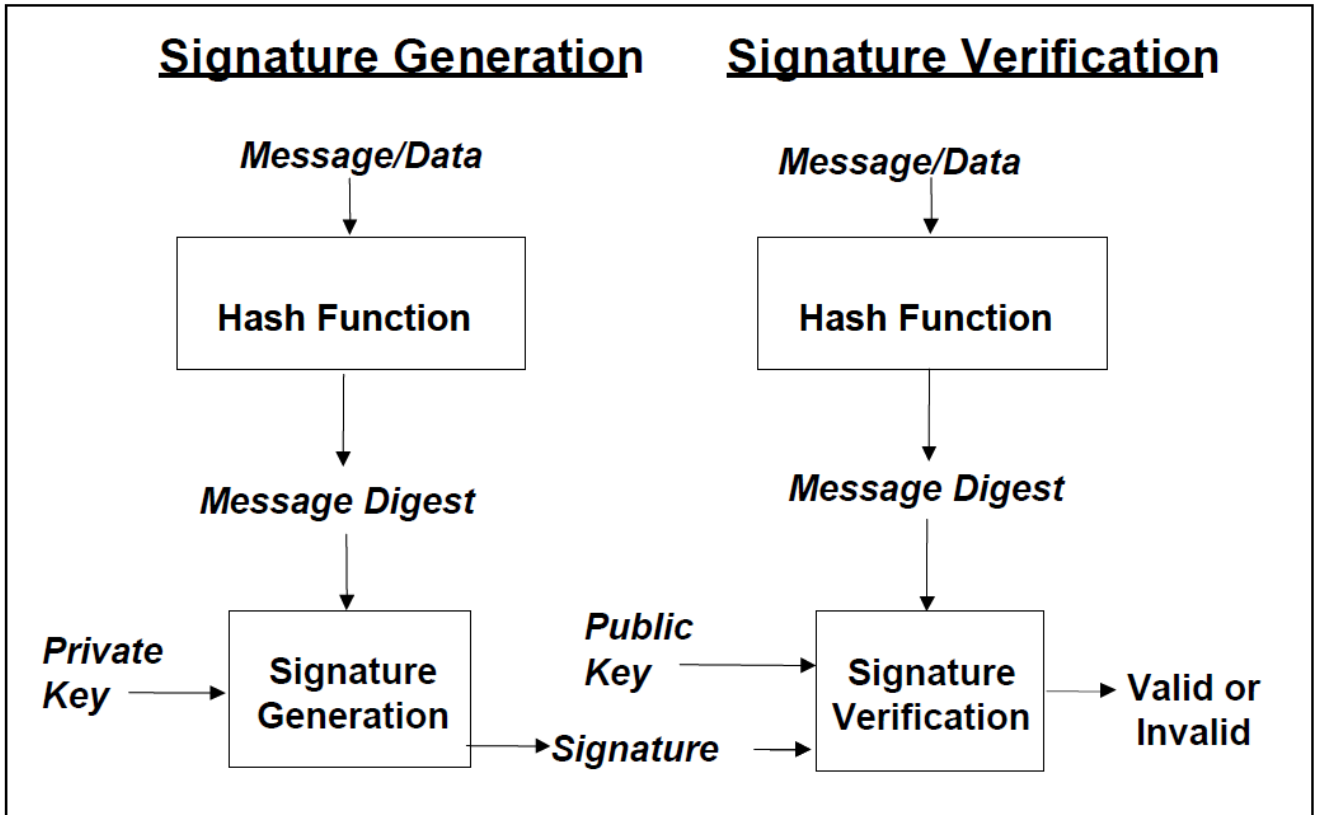


Figure 1: Digital Signature Processes

DSS

- Selection of Parameter Sizes and Hash Functions

- Domain Parameter Generation
  - only for DSA, ECDSA
- Signature Generation

- Signature Verification and Validation

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]

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.

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]