095946- ADVANCED ALGORITHMS AND PARALLEL PROGRAMMING

Fabrizio Ferrandi

a.a. 2021-2022

Primality test

p è divisibile solo per se stesso e per 1

Definition:

An integer $p \geq 2$ is prime **iff** ($a \mid p \rightarrow a = 1$ **or** $a = p$).

Algorithm: deterministic primality test (naive)

Input: integer $n \geq 2$

Output: answer to the question: Is n prime?

```
if n = 2 then return true
if n even then return false
for i = 1 to √n/2 do
      if 2i + 1 divides n
      then return false
return true
```

Complexity: $\Theta(\sqrt{n})$

Primality test

Goal:

Randomized method

- Polynomial time complexity (in the length of the input)
- If answer is "not prime", then *n* is not prime
- If answer is "prime", then the probability that *n* is not prime is at most $p>0$, *n is composite*

k iterations: probability that *n* is not prime is at most p^k

Example of one-sided error Monte Carlo Algorithm: false-biased

Primality test

Observation:

Each odd prime number p divides $2^{p-1} - 1$.

from Fermat

Examples: $p = 17$, $2^{16} - 1 = 65535 = 17 * 3855$

$p = 23$, $2^{22} - 1 = 4194303 = 23 * 182361$

Simple primality test:

1 Calculate $z = 2^{n-1} \bmod n$

2 if $z = 1$

3 then n is possibly prime

4 else n is composite

Advantage: This only takes polynomial time

Simple primality test

Definition:

A natural number $n>=2$ is a base-2 pseudoprime if n is composite and

$$2^{n-1} \bmod n = 1.$$

Example: $n = 11 * 31 = 341$

$$2^{340} \bmod 341 = 1$$

Randomized primality test

Theorem: (Fermat's little theorem)

If p prime and $0 < a < p$, then

$$a^{p-1} \bmod p = 1.$$

Definition:

n is pseudoprime to base a, if n not prime and

$$a^{n-1} \bmod n = 1.$$

Example: $n = 341, \; a = 3$

$$3^{340} \bmod 341 = 56 \neq 1$$

Randomized primality test

Algorithm: Randomized primality test 1

1 Randomly choose $a \in [2, n\text{-}1]$
2 Calculate a^{n-1} mod n
3 **if** a^{n-1} mod n = 1
4 **then** n is possibly prime
5 **else** n is composite

Prob(n is not prim, but a^{n-1} mod n = 1) ?

Carmichael numbers

Problem: Carmichael numbers

Definition: A number n>=2 is a Carmichael number if n is composite and for any a with GCD(a, n) = 1 we have

$$a^{n-1} \bmod n = 1$$

Example:

Smallest Carmichael number: 561 = 3 * 11 * 17

Randomized primality test 2

Theorem:

If p prime and $0 < a < p$, then the only solutions to the equation

$$a^2 \bmod p = 1$$

are $a = 1$ and $a = p - 1$.

Definition:

a is called non-trivial square root of 1 mod n, if

$$a^2 \bmod n = 1 \quad \text{and} \quad a \neq 1, n - 1.$$

Example: $n = 35$

$$6^2 \bmod 35 = 1$$

Fast exponentiation

Idea:

During the computation of a^{n-1} ($0 < a < n$ randomly chosen), test whether there is a non-trivial square root mod n.

Method for the computation of a^n:

Case 1: [n is even]

$$a^n = a^{n/2} * a^{n/2}$$

Case 2: [n is odd]

$$a^n = a^{(n-1)/2} * a^{(n-1)/2} * a$$

Fast exponentiation

Example:

$a^{62} = (a^{31})^2$

$a^{31} = (a^{15})^2 * a$

$a^{15} = (a^{7})^2 * a$

$a^{7} = (a^{3})^2 * a$

$a^{3} = (a)^2 * a$

Complexity: $O(\log n)$

Fast exponentiation

```
boolean isProbablyPrime;

power(int a, int p, int n) {
    /* computes a^p mod n and checks during the
       computation whether there is an x with
       x^2 mod n = 1 and x ≠ 1, n-1 */

    if (p == 0) return 1;
    x = power(a, p/2, n)
    result = (x * x) % n;
```

Fast exponentiation

```
 /* check whether x^2 mod n = 1 and x ≠ 1, n-1 */
    if (result == 1 && x != 1 && x != n -1 )
        isProbablyPrime = false;

    if (p % 2 == 1)
        result = (a * result) % n;

    return result;
}
```

Complexity: $O(\log^2 n \log p)$

Randomized primality test 2

```
primalityTest(int n) {
    /* executes the randomized primality test for a
   chosen at random*/

    a = random(2, n-1);

    isProbablyPrime = true;

    result = power(a, n-1, n);

    if (result != 1 || !isProbablyPrime)
        return false;
    else
        return true;
}
```

Randomized primality test 2

Theorem:

If n is composite, there are at most

$$n\text{-}9/4$$

integers $0 < a < n$, for which the algorithm primalityTest fails.

Application

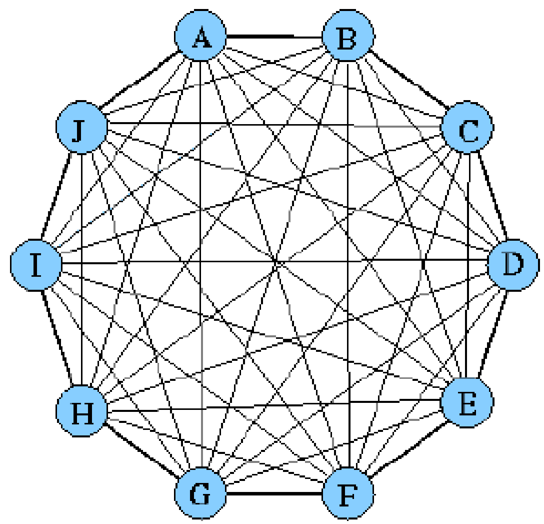
- Public-Key Cryptosystems

Secret key cryptosystems

Traditional encryption of messages with secret keys

Disadvantages:

1. The key *k* has to be exchanged between A and B before the transmission of the message.
2. For messages between *n* parties $n(n\text{-}1)/2$ keys are required.



Advantage:

Encryption and decryption can be computed very efficiently.

Electronic security services

- **Guarantees:**
- • Confidentiality of the transmission
- • Integrity of the data
- • Authenticity of the sender
- • Liability of the transmission

Public-key cryptosystems

Diffie and Hellman (1976)

Idea: Each participant A has two keys:

1. a public key P_A accessible to every other participant
2. a private (or: secret) key S_A only known to A.

Public-key cryptosystems

D = set of all legal messages,
e.g. the set of all bit strings of finite length

$$P_A, S_A : D \rightarrow D$$

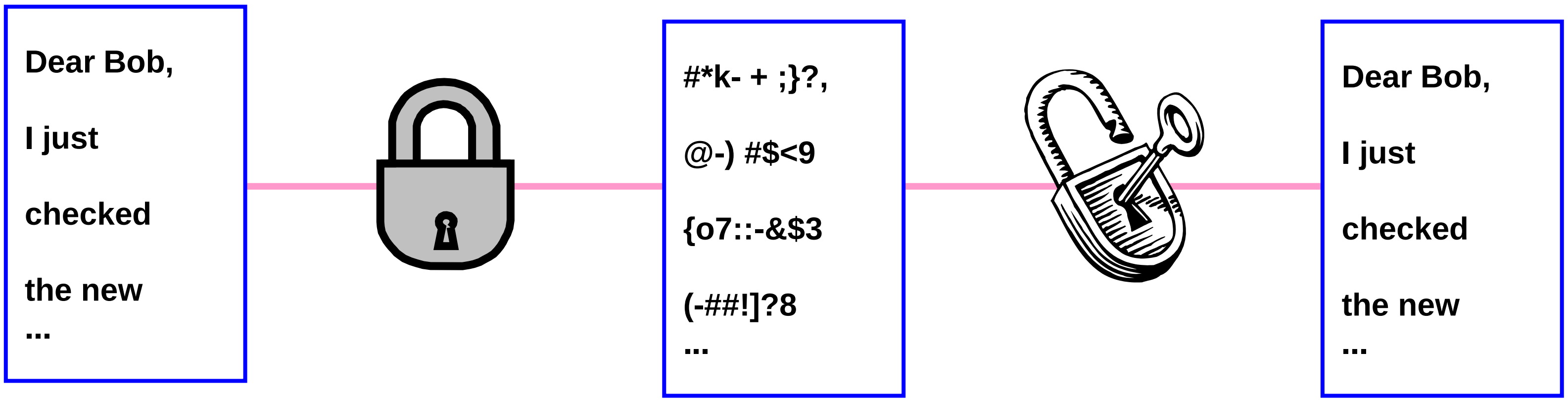
Sa means "secret key of a"

Three conditions:

1. P_A and S_A can be computed efficiently
2. $S_A(P_A(M)) = M$ and $P_A(S_A(M)) = M$
   (P_A, S_A are inverse functions) inverse because they're related to encryption and decryption
3. S_A is not computable from P_A (with realistic effort)

Encryption in a public-key system

A sends a message *M* to *B*.



Encryption in a public-key system

1. *A* accesses *B*'s public key P_B (from a public directory or directly from *B*).

2. *A* computes the encrypted message $C = P_B(M)$ and sends C *to* **B**.

3. After **B** has received message C, **B** decrypts the message with his own private key S_B: $M = S_B(C)$

Generating a digital signature

A sends a digitally signed message M' to *B*:

1. *A* computes the digital signature σ for M' with her/his own private key:

$$\sigma = S_A(M')$$

2. *A* sends the pair (M',σ) to *B*.

3. After receiving (M',σ), *B* checks the digital signature:

$$P_A(\sigma) = M'$$

Anybody is able to check σ using P_A (e.g. for bank checks).

RSA cryptosystems

R. Rivest, A. Shamir, L. Adleman

Generating the public and private keys:

1. Randomly select two primes p and q of similar size, each with $l+1$ bits ($l \geq 500$).

2. Let $n = p \cdot q$

3. Let e be an integer that does not divide $(p - 1) \cdot (q - 1)$.

   i.e. e relatively prime to $(p - 1) \cdot (q - 1)$ -> GCD(e, $(p - 1) \cdot (q - 1)$ ≡ 1

4. Calculate $d = e^{-1} \bmod (p - 1)(q - 1)$

   i.e.: $d \cdot e \equiv 1 \bmod (p - 1)(q - 1)$

RSA cryptosystems

5. Publish $P = (e, n)$ as public key

6. Keep $S = (d, n)$ as private key

Divide message (represented in binary) in blocks of size $2 \cdot l$.
Interpret each block M as a binary number: $0 \leq M < 2^{2 \cdot l}$

$$P(M) = M^e \bmod n \qquad\qquad S(C) = C^d \bmod n$$

Multiplicative inverse

Algorithm: extended-Euclid

Input: Two integers a and b where $b \geq 0$

Output: gcd(a,b) and two integers x and y with

$$xa + yb = \gcd(a,b)$$

if $b = 0$ then return $(a, 1, 0)$;

(d, x', y') := extended-Euclid(b, a mod b);

$x := y'$; $y := x' - \lfloor a/b \rfloor y'$;

return (d, x, y);

Application: $a=(p-1)(q-1)$ $b=e$

Integers x and y with

$$x(p-1)(q-1) + ye = \gcd((p-1)(q-1),e) = 1$$

Acknowledge

- Based on Randomized Algorithms, by R. Motwani and P. Raghavan.
- Based on Lectures of Prof. Dr. Th. Ottmann and of Prof. Dr. Susanne Albers: http://electures.informatik.uni-freiburg.de/portal/web/guest