4. Primality test

Definition:

A natural number $p \geq 2$ is prime iff $a \mid p$ implies that $a = 1$ or $a = p$.

We consider primality tests for numbers $n \geq 2$.

Algorithm: Deterministic primality test (naive approach)

Check all numbers
$a = 1, \ldots, \sqrt{n}$

Input: Natural number $n \geq 2$

Output: Answer to the question „Is n prime?"

if n = 2 **then return** true
if n even **then return** false
for i = 1 **to** $\sqrt{n}\,/\,2$ **do**
  if $2i + 1$ divides n
  then return false
return true

Input size $\Theta(\log n)$
Running time $O(2^{\log n / 2})$
not polynomial running time

Running time: $\Theta(\sqrt{n})$

Primality test

Goal: Monte Carlo

Randomized algorithm

- Polynomial running time.
- If it returns "not prime", then *n* is not prime.
- If it returns "prime", then with probability at most *p, p>0, n* is composite.

Later $p \approx \frac{1}{4}$

After *k* iterations: with probability p^k, *n* is composite. even though reported "prime"

Primality test

Fact: For any odd prime number p: $2^{p-1} \bmod p = 1$.

$2^{16} = 65535+1$

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

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

$2^{16} \bmod 17 = 1$

Deterministic: Input n

Simple primality test:

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

2 **if** $z = 1$

3 **then** n is possibly prime

4 **else** n is composite

Repeated Squaring a^n

$$pow(a, n) = \begin{cases} 1 & n = 0 \\ a \cdot pow(a, n-1) & n \text{ odd} \\ \left[pow(a, \frac{n}{2})\right]^2 & n \text{ even} \end{cases}$$

$$a^n = \left(a^{\frac{n}{2}}\right)^2$$

Advantage: polynomial running time.

Simple primality test

Definition:

A natural number $n \geq 2$ is a base-2 pseudoprime if n is composite and

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

Example: $n = 11 * 31 = 341$ is a base-2 pseudoprime

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

Randomized primality test

Theorem: (Fermat's little theorem)
If p is prime and $0 < a < p$, then

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

Example: $n = 341$, $a = 3$: $3^{340} \bmod 341 = 56 \neq 1$

Algorithm: Randomized primality test 1

1 Choose a in the range $[2, n\text{-}1]$ uniformly at random
2 Compute $a^{n-1} \bmod n$
3 **if** $a^{n-1} \bmod n = 1$
4 **then** n is probably prime
5 **else** n is composite

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

Prob(n is composite but $a^{n-1} \bmod n = 1$) ?

For Carmichael numbers this probability could be quite large

Problem: Carmichael numbers

Definition:

A natural number $n \geq 2$ is a base-a pseudoprime if n is composite and

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

Definition: A number $n \geq 2$ 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

$a^2 \bmod p = 1$

$a^2 = 1 + k \cdot p \qquad k \in \mathbb{N}_0$

$a^2 - 1 = k \cdot p$

$(a-1) \cdot (a+1) = k \cdot p$

(1) $a - 1 = p \Rightarrow a = p+1$ not possible

(2) $a + 1 = p \Rightarrow a = p-1$

(3) $a = 1 \qquad k = 0$

Theorem: If p is prime and $0 < a < p$, then the equation

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

has exactly the two solutions $a = 1$ and $a = p - 1$.

Definition: A number a is a non-trivial square root mod n if

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

Example: $n = 35$

$6^2 \bmod 35 = 1$

$36 \bmod 35 = 1$

$6 \neq 1, \quad 6 \neq 34$

Fast exponentiation

Idea:

While computing a^{n-1}, where $0 < a < n$ is chosen uniformly at random, check if a non-trivial square root mod n exists.

Method for computating a^n: Repeated Squaring

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:

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

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

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

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

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

Running time: $O(log^2 a^n \; log \; n)$

Fast exponentiation

```
boolean isProbablyPrime;
```

global variable, initially true

```
power(int a, int p, int n){
```

Computes $a^p \bmod n$

/* computes a^p mod n and checks if a number x with x^2 mod n = 1 and $x \neq 1, n-1$ occurs during the computation */

```
    if (p == 0) return 1;

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

$x = a^{\lfloor \frac{p}{2} \rfloor} \bmod n$

$result = x^2 \bmod n$

Check if x is a non-trivial square root

Fast exponentiation

```
/* check if x² 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;
}
```

x in n.t.s.r

result $= \left(a^{\lfloor \frac{p}{2} \rfloor}\right)^2$

Running time: $O(\log^2 n \log p)$

polynomial !

Randomized primality test 2

```
primeTest(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;
}
```

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

n composite

Randomized primality test 2

Theorem:

If n is composite, then there are at most

$$\frac{n-9}{4} \approx \frac{n}{4}$$

numbers $0 < a < n$, for which the algorithm primeTest fails.

$\Rightarrow$ Error probability of primeTest is $< \frac{1}{4}$

$\Rightarrow$ After k independent repetitions we have overall ~~error~~ error probability $< \left(\frac{1}{4}\right)^k$

5. Application

Public-Key Cryptosystems

Secret key cryptosystems

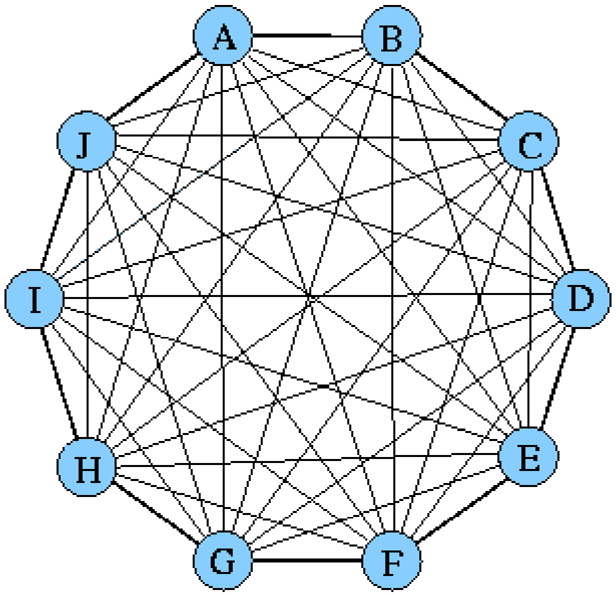
Parties want to exchange secure messages

Traditional encryption of messages

Disadvantages:

1. Prior to transmission of the message, the key *k* has to be exchanged between the parties A und B.
2. For encryption of messages between *n* parties, $n(n-1)/2$ keys are required.

$\Theta(n^2)$



large number of keys need exchange via a safe channel

Secret key encryption systems

Advantage:

Encryption and decryption are fast.

Electronic security services

Guarantees:

- Confidentiality of the transmission
- Integrity of the data
- Authenticity of the sender
- Liability of the transmission

Public-key cryptosystems

Framework

Diffie and Hellman (1976)

Idea: Each participant *A* holds two keys:

1. A public key P_A , accessible to all other participants.
2. A private key S_A that is kept secret.

Public-key cryptosystems

D = Set of all valid messages,
e.g. set of all bitstrings of finite length

$D = \{0,1\}^*$

$$P_A(\ ), S_A(\ ) : D \xrightarrow{1-1} D$$

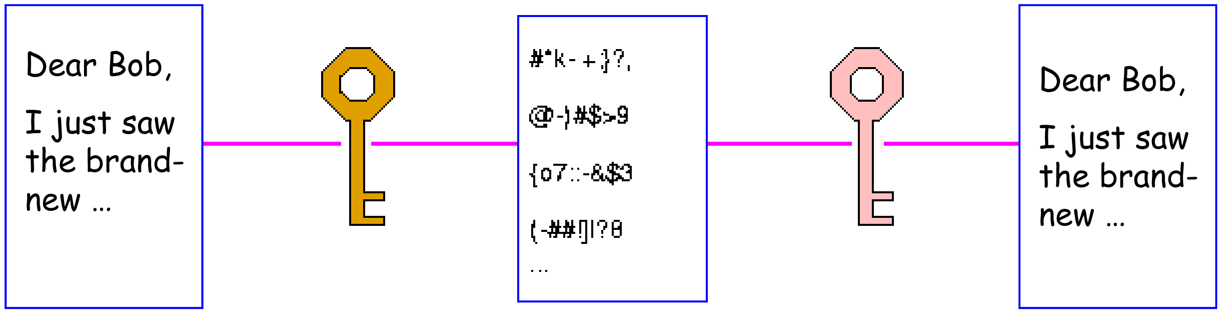
Three constraints:

1. $P_A()$, $S_A()$ efficiently computable — poly. time
2. $S_A(P_A(M)) = M$ and $P_A(S_A(M)) = M$ — correctness
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*:

Alice Bob



M C

A downloads P_B

A computes $P_B(M) = C$ ciphertext

A sends C to B

B computes $S_B(C) = S_B(P_B(M)) = M$

RSA crypto system