Theoretical foundations

Introduction to Computational Number Theory - Part 6

Jean-Sébastien Coron

Université du Luxembourg

October 25, 2009

Summary

- C programming
  - Structures.
  - Pointers and structures.
  - Arrays of structures.
- Algorithmic number theory
  - Modular exponentiation

Structures

- Using structures

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d p;

p.x=2;
p.y=3;
Point2d q=p;
printf("%f\n",q.x);
```

Pointers and structures

- A pointer can refer to a structure.

```
typedef struct {
  float x;
  float y;
} Point2d;
Point2d p;
Point2d *q;
p.x=5;
q=&p;
(*q).y=3;
q->y=3; // equivalent
printf("%f\n",q->x);
printf("%f\n",p.y);
```

- Allocating memory :

```
typedef struct {
 float x;
 float y;
} Point2d;

Point2d *p;
p=(Point2d *) malloc(sizeof(Point2d));

p->x=3;
p->y=p->x+2;;
printf("%f\n",p->y);
```

Array of structures

- One can define an array of structures :

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d t[10];

t[5].x=3;
t[7].y=5;
```

Dynamic array

- One can define a dynamic array of structures :

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d *t;

t=(Point2d *) malloc(10*sizeof(Point2d));

t[5].x=3;
t[7].y=5;
```

Modular exponentiation

- We want to compute $c = a^b \mod n$.
  - Example: RSA
    - $c = m^e \mod N$ where m is the message, e the public exponent, and N the modulus.
- Naïve method:
  - Multiplying a in total b times by itself modulo n
  - Very slow: if b is 100 bits, roughly 2^{100} multiplications !

Square and multiply algorithm

- Let $b = (b_{\ell-1} \dots b_0)_2$ the binary representation of b
  - $b = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$
- Square and multiply algorithm :
  - Input : a, b and n
  - Output : $a^b \mod n$
  - $c \leftarrow 1$

    for $i = \ell - 1$ down to 0 do

    $\quad c \leftarrow c^2 \mod n$

    $\quad$ if $b_i = 1$ then $c \leftarrow c \cdot a \mod n$

    Output c

Analyis

- Let B_i be the integer with binary representation $(b_{\ell-1} \ldots b_i)_2$
  - $B_i = \sum_{j=i}^{\ell-1} b_j \cdot 2^{j-i}$
  - $B_{i-1} = 2 \cdot B_i + b_{i-1}$
- Claim : let c_i be the value of c at the end of step i :

$$c_i = a^{B_i} \mod n$$

  - Claim is true for $i = \ell - 1$
    - $B_{\ell-1} = b_{\ell-1}$
    - $c_{\ell-1} = 1$ if $b_{\ell-1} = 0$ and $c_{\ell-1} = a$ if $b_{\ell-1} = 1$
    - $c_{\ell-1} = a^{b_{\ell-1}} = a^{B_{\ell-1}} \mod n$

Analysis (2)

- Assume that claim is true for i.
  - Then $c_i = a^{B_i} \mod n$
  - $c_{i-1} = (c_i)^2 \mod n$ if $b_{i-1} = 0$
  - $c_{i-1} = (c_i)^2 \cdot a \mod n$ if $b_{i-1} = 1$

$$\begin{aligned} c_{i-1} &= (c_i)^2 \cdot a^{b_{i-1}} \mod n \\ c_{i-1} &= (a^{B_i})^2 \cdot a^{b_{i-1}} \mod n \\ c_{i-1} &= a^{2 \cdot B_i + b_{i-1}} = a^{B_{i-1}} \mod n \end{aligned}$$

- The output value c is $c = c_0$
  - $c_0 = a^{B_0} \mod n$ and $B_0 = b$ gives

$$c = a^b \mod n$$