Algorithms for Numbers and Public-key cryptography

Part 2

Jean-Sébastien Coron

Université du Luxembourg

March 7, 2018

Summary

- Computing with large integers.
  - Addition, multiplication.
  - Division with reminder
  - Modular exponentiation
  - Probabilistic primality testing

Computing with large integers

- Limited precision in C :
  - `int`: 32 bits. Computing with values $< 2^{32}$.
- Computing with large integers :
  - One represents the big integers in base B in an array.
  - One implements addition, multiplication, division on big integers.
  - Existing libraries :
    - GMP: `www.swox.com/gmp`
    - NTL: `www.shoup.net`
    - Some parts written in assembly for better efficiency.

Technique

- Representing large integers :
  - An integer is represented as an array of digits in base B, with a sign bit.

$$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm (a_{k-1} \dots a_0)_B$$

    with $0 \leq a_i < B$. If $a \neq 0$, we must have $a_{k-1} \neq 0$.
- Basis :
  - One generally takes $B = 2^v$ for some v.
  - One can also take $B = 10$.

Addition

- Computing $c = a + b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Ket $c = (c_k c_{k-1} \dots c_0)$

$carry \leftarrow 0$
for $i = 0$ to $\ell - 1$ do
  $tmp \leftarrow a_i + b_i + carry$
  $carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
for $i = \ell$ to $k - 1$ do
  $tmp \leftarrow a_i + carry$
  $carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
$c_k \leftarrow carry$

Substraction

- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
      $tmp \leftarrow a_i - b_i + carry$
      $carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
    for $i = \ell$ to $k - 1$ do
      $tmp \leftarrow a_i + carry$
      $carry \leftarrow tmp/B$; $c_i \leftarrow tmp \mod B$
    $c_k \leftarrow carry$
  - If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and let $c := -c'$.

Multiplication

- Computing $c = a \cdot b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ avec $k, \ell \geq 1$. Let $c = (c_{k+\ell-1} \dots c_0)$

$carry \leftarrow 0$
for $i = 0$ to $k + \ell - 1$ do
  $c_i \leftarrow 0$
for $i = 0$ to $k - 1$ do
  $carry \leftarrow 0$
  for $j = 0$ to $\ell - 1$ do
    $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
    $carry \leftarrow tmp/B$; $c_{i+j} \leftarrow tmp \mod B$
  $c_{i+\ell} \leftarrow carry$

Division with remainder

- Let $a = (a_{k-1} \dots a_0)_B$ and $b = (b_{\ell-1} \dots b_0)_B$ with $a > b > 0$ and $b_{\ell-1} \neq 0$.
  - Compute q and r such that $a = b \cdot q + r$ and $0 \leq r < b$.
  - $q = (q_{m-1} \dots q_0)_B$, with $m := k - \ell + 1$.
- Algorithm overview:

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow r/(B^i b)$
  $r \leftarrow r - B^i \cdot q_i \cdot b$
output r

Division with remainder

- For all i, $0 \leq r < B^i \cdot b$ after step i
  - Therefore, $0 \leq r < b$ eventually.
- How to compute $q_i = r/(B^i \cdot b)$
  - Test all possible values of $0 \leq q_i < B$
  - Not efficient, except if B is small (*e.g.* $B = 10$).
  - Possible to do much better

Division with remainder

- Complete algorithm (for small B)

$r \leftarrow a$
for $i = m - 1$ downto 0 do
  $q_i \leftarrow 0$
  while $r >= 0$
    $r \leftarrow r - B^i \cdot b$
    $q_i \leftarrow q_i + 1$
  $q_i \leftarrow q_i - 1$
  $r \leftarrow r + B^i \cdot b$
output r

Summary

- For $a \in \mathbb{Z}$, let $\mathrm{len}(a)$ be the number of bits in the binary representation of $|a|$:
  - $\mathrm{len}(a) = \lfloor \log_2 |a| \rfloor + 1$ if $a \neq 0$
  - $\mathrm{len}(0) = 1$
- Let a and b be two arbitrary integers
  - We can compute $a \pm b$ in time $\mathcal{O}(\mathrm{len}(a) + \mathrm{len}(b))$
  - We can compute $a \cdot b$ in time $\mathcal{O}(\mathrm{len}(a)\,\mathrm{len}(b))$
  - If $b \neq 0$, we can compute the quotient q and the remainder r in $a = b \cdot q + r$ in time $\mathcal{O}(\mathrm{len}(b)\,\mathrm{len}(q))$

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

      $c \leftarrow c^2 \mod n$

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

    Output c

Analysis

- Let B_i be the integer with binary representation $(b_{\ell-1} \dots b_i)_2$
  - $B_i = \sum_{j=i}^{\ell-1} b_j \cdot 2^{j-i}$
  - $B_{i-1} = 2 \cdot B_i + b_{i-1}$
- 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

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

Primality Testing

- Motivation for prime generation:
  - Generate the primes p and q in RSA.
  - p and q must be large: at least 512 bits.
- Goal of primality testing:
  - Given an integer n, determine whether n is prime or composite.
- Simplest algorithm: trial division.
  - Test if n is divisible by 2, 3, 4, 5,... We can stop at $\sqrt{n}$.
  - Algorithm determines if n is prime or composite, and outputs the factors of n if n is composite.
  - Very inefficient algorithm
    - Requires around $\sqrt{n}$ arithmetic operations.
    - If n has 256 bits, then 2^{128} arithmetic operations. If 2^{30} operations/s, this takes 10^{22} years !

Probabilistic primality testing

- Goal: describe an efficient probabilistic primality test.
  - Can test primality for a 512-bit integer n in less than a second.
- Probabilistic primality testing.
  - The algorithm does not find the factors of n.
  - The algorithm may make a mistake (pretend that an integer n is prime whereas it is composite).
  - But the mistake can be made arbitrarily small (*e.g.* $< 2^{-100}$, so this makes no difference in practice.

Distribution of prime numbers

- Let $\pi(x)$ be the number of primes in the interval $[2, x]$.
- Theorem (Prime number theorem)
  - We have $\pi(x) \simeq x/\log x$.
- Fact (approximation of the n-th prime number)
  - Let p_n denote the n-th prime number. Then $p_n \simeq n \cdot \log n$. More explicitely,

$$n \log n < p_n < n(\log n + \log\log n) \quad \text{for } n \geq 6$$

The Fermat test

- Fermat's little theorem
  - If n is prime and a is an integer between 1 and $n-1$, then $a^{n-1} \equiv 1 \mod n$.
  - Therefore, if the primality of n is unknown, finding $a \in [1, n-1]$ such that $a^{n-1} \not\equiv 1 \mod n$ proves that n is composite.
- Fermat primality test with security parameter t.

For $i = 1$ to t do
  Choose a random $a \in [2, n-2]$
  Compute $r = a^{n-1} \mod n$
  If $r \neq 1$ then return "composite"
Return "prime'

- Complexity: $\mathcal{O}(t \cdot \log^3 n)$

Analysis of Fermat's test

- Let $L_n = \{a \in [1, n-1] : a^{n-1} \equiv 1 \mod n\}$
- Theorem:
  - If n is prime, then $L_n = \mathbb{Z}_n^*$. If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$, then $|L_n| \leq (n-1)/2$.
- Proof:
  - If n is prime, $L_n = \mathbb{Z}_n^*$ from Fermat.
  - If n is composite, since L_n is a sub-group of $\mathbb{Z}_n^*$ and the order of a subgroup divides the order of the group, $|\mathbb{Z}_n^*| = m \cdot |L_n|$ for some integer m.

$$|L_n| = \frac{1}{m}|\mathbb{Z}_n^*| \leq \frac{1}{2}|\mathbb{Z}_n^*| \leq \frac{n-1}{2}$$

Analysis of Fermat's test

- If n is composite and $L_n \subsetneq \mathbb{Z}_n^*$
  - then $a^{n-1} = 1 \mod n$ with probability at most $1/2$ for a random $a \in [2, n-2]$.
  - The algorithm outputs "prime" wih probability at most 2^{-t}.
- Unfortunately, there are odd composite numbers n such that $L_n = \mathbb{Z}_n^*$.
  - Such numbers are called Carmichael numbers. The smallest Carmichael number is 561.
  - Carmichael numbers are rare, but there are an infinite number of them, so we cannot ignore them.

The Miller-Rabin test

- The Miller-Rabin test is based on the following fact:
  - Let n be a prime > 2, let $n - 1 = 2^s \cdot r$ where r is odd. Let a be any integer such that $\gcd(a, n) = 1$. Then either $a^r \equiv 1 \mod n$ or $a^{2^j \cdot r} \equiv -1 \mod n$ for some j, $0 \leq j \leq s - 1$.
- Proof:
  - Since n is prime, $a^{n-1} \equiv 1 \mod n$, therefore $a^{r \cdot 2^s} \equiv 1 \mod n$
  - Consider j_0 the minimum $0 \leq j \leq s - 1$ such that $a^{r \cdot 2^{j+1}} \equiv 1 \mod n$. Let $\beta := a^{r \cdot 2^{j_0}} \mod n$
  - Then $\beta^2 \equiv 1 \mod n$. We must have $\beta = \pm 1$ because a polynomial of degree 2 has at most two roots over $\mathbb{Z}_n$ for n prime.
  - If $j_0 = 0$, then $a^r \equiv \pm 1 \mod n$
  - If $j_0 > 0$, then we must have $a^{r 2^{j_0}} \equiv -1 \mod n$ (instead j_0 would not be the minimum).

The Miller-Rabin test

Write $n-1=2^s \cdot r$ for odd r.
For $i=1$ to t do
  Generate a random $a \in [2, n-2]$. Let $\beta \leftarrow a^r \mod n$.
  If $\beta \neq 1$ and $\beta \neq -1$ do
    $j \leftarrow 1$.
    While $j \leq s-1$ and $\beta \neq -1$ do
      Let $\beta \leftarrow \beta^2 \mod n$
      If $\beta = +1$ return "composite"
      $j \leftarrow j+1$
    If $\beta \neq -1$ return "composite"
Return "prime"

The Miller-Rabin test

- Property
  - If n is prime, then the Miller-Rabin test always declares n as prime.
  - If $n \geq 3$ is composite, then the probability that the Miller-Rabin test outputs "prime" is less than $\left(\frac{1}{4}\right)^t$
- Most widely used test in practice.
  - With $t = 40$, error probabitility less than 2^{-80}. Much less than the probability of a hardware failure.
  - Can test the primality of a 512-bit integer in less than a second.
  - Complexity: $\mathcal{O}(\log^3 n)$

Prime number generation

- To generate a prime integer of size ℓ bits
  - Generate a random integer n of size ℓ bits
  - Test its primality with Miller-Rabin.
  - If n is declared prime, output n, otherwise generate another n again.