Computer and Network Security

MIT 6.857 Class Notes

by Ronald L. Rivest
December 2, 2002

Introduction to Number Theory

Elementary number theory provides a rich set of tools for the implementation of cryptographic schemes. Most public-key cryptosystems are based in one way or another on number-theoretic ideas.

The next pages provide a brief introduction to some basic principles of elementary number theory.

Bignum computations

Many cryptographic schemes, such as RSA, work with large integers, also known as "bignums" or "multi-precision integers." Here "large" may mean 160–4096 bits (49–1233 decimal digits), with 1024-bit integers (308 decimal digits) typical. We briefly overview of some implementation issues and possibilities.

When RSA was invented, efficiently implementing it was a problem. Today, standard desktop CPU's perform bignum computations quickly. Still, for servers doing hundreds of SSL connections per second, a hardware assist may be needed, such as the SSL accelerators produced by nCipher `www.ncipher.com/`.

A popular C/C++ software subroutine library supporting multi-precision operations is **GMP** (GNU Multi-precision package) `www.swox.com/gmp/`. A more elaborate package (based on GMP) is Shoup's NTL (Number Theory Library) `www.shoup.net/ntl/`. For a survey, see `https://www.cosic.esat.kuleuven.ac.be/nessie/call/mplibs.html`.

Java has excellent support for multiprecision operations in its BigInteger class `java.sun.com/j2se/1.4.1/docs/api/java/math/BigInteger.html`; this includes a primality-testing routine.

Python `www.python.org/` is a personal favorite; it includes direct support for large integers.

Scheme `www.swiss.ai.mit.edu/projects/scheme/` also provides direct bignum support.

Some other pointers to software and hardware implementations can be found in the "Practical Aspects" section of Helger Lipmaa's "Cryptology pointers" `www.tcs.hut.fi/˜helger/crypto/=`.

When working on k-bit integers, most implementations implement addition and subtraction in time $O(k)$, multiplication, division, and gcd in time $O(k^2)$ (although faster implementations exist for very large k), and modular exponentation in time $O(k^3)$.

To get you roughly calibrated, here are some timings, obtained from a simple Python program on my IBM Thinkpad laptop (1.2 GHz PIII processor) on 1024-bit inputs. SHA-1 is included just for comparison. The last column gives the approximate ratio of running time to addition.

2.2 microseconds	addition	455,000 per second	1
4.4 microseconds	SHA1 hash (on 20-byte input)	227,000 per second	2
10.8 microseconds	modular addition	93,000 per second	5
41 microseconds	multiplication	24,000 per second	20
135 microseconds	modular multiplication	7,400 per second	60
2.3 milliseconds	modular exponentiation (exponent is 2**16+1)	440 per second	1000
5.5 milliseconds	gcd	180 per second	2500
204 milliseconds	modular exponentiation (1024-bit exponent)	5 per second	93000

Divisors and Divisibility

Definition 1 (Divides relation, divisor, common divisor) *We say that "d divides a", written $d \mid a$, if there exists an integer k such that $a = kd$. If d does not divide a, we write "$d \not| a$". If $d \mid a$ and $d \geq 0$, we say that d is a* divisor *of a. If $d \mid a$ and $d \mid b$, then d is a* common divisor *of a and b.*

Example 1 *Every integer $d \geq 0$ (including $d = 0$) is a divisor of 0. While 0 divides no integer except itself, 1 is a divisor of every integer. The divisors of 12 are $\{1, 2, 3, 4, 6, 12\}$. A common divisor of 14 and 77 is 7. If $d \mid a$ then $d \mid (-a)$.*

Definition 2 (prime) *An integer $p > 1$ is* prime *if its only divisors are 1 and p.*

Definition 3 (Greatest common divisor, relatively prime) *The* greatest common divisor, $\gcd(a, b)$, *of two integers a and b is the largest of their common divisors, except that $\gcd(0, 0) = 0$ by definition. Integers a and b are* relatively prime *if $gcd(a, b) = 1$.*

Example 2

$$\begin{aligned} \gcd(24, 30) &= 6 \\ \gcd(4, 7) &= 1 \\ \gcd(0, 5) &= 5 \\ \gcd(-6, 10) &= 2 \end{aligned}$$

Example 3 *For all $a \geq 0$, a and $a + 1$ are relatively prime. The integer 1 is relatively prime to all other integers.*

Example 4 *If p is prime and $1 \leq a < p$, then $\gcd(a, p) = 1$. That is, a and p are relatively prime.*

Definition 4 *For any positive integer n, we define* Euler's phi function of n, *denoted $\phi(n)$, as the number of integers d, $1 \leq d \leq n$, that are relatively prime to n. (Note that $\phi(1) = 1$.)*

Example 5 *If p is prime, then $\phi(p) = p - 1$. For any integer $k > 0$, $\phi(2^k) = 2^{k-1}$.*

Definition 5 *The* least common multiple $\operatorname{lcm}(a, b)$ *of two integers $a \geq 0$, $b \geq 0$, is the least m such that $a \mid m$ and $b \mid m$.*

Exercise 1 *Show that the number of divisors of $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ (where the p_i's are distinct primes) is $\prod_{1 \leq i \leq k}(1 + e_i)$.*

Exercise 2 *Show that $\operatorname{lcm}(a, b) = ab / \gcd(a, b)$.*

Fermat's Little Theorem

Theorem 1 (Fermat's Little Theorem) *If p is prime and $a \in \mathbf{Z}_p^*$, then $a^{p-1} = 1 \pmod p$.*

Theorem 2 (Lagrange's Theorem) *The order of a subgroup must divide the order of a group.*

Fermat's Little Theorem follows from Lagrange's Theorem, since the order of the subgroup $\langle a \rangle$ generated by a in $\mathbf{Z}_p^*$ is the least $t > 0$ such that $a^t = 1 \pmod p$, and $|\mathbf{Z}_p^*| = p - 1$.

Euler's Theorem generalizes Fermat's Little Theorem, since $|\mathbf{Z}_n^*| = \phi(n)$ for all $n > 0$.

Theorem 3 (Euler's Theorem) *For any $n > 1$ and any $a \in \mathbf{Z}_n^*$, $a^{\phi(n)} = 1 \pmod n$.*

A somewhat tighter result actually holds. Define for $n > 0$ Carmichael's lambda function $\lambda(n)$ to be the least positive t such that $a^t = 1 \pmod n$ for all $a \in \mathbf{Z}_n^*$. Then $\lambda(1) = \lambda(2) = 1$, $\lambda(4) = 2$, $\lambda(2^e) = 2^{e-2}$ for $e > 2$, $\lambda(p^e) = p^{e-1}(p-1)$ if p is an odd prime, and if $n = p_1^{e_1} \cdots p_k^{e_k}$, then

$$\lambda(n) = \mathrm{lcm}(\lambda(p_1^{e_1}), \ldots, \lambda(p_k^{e_k})) \ .$$

Computing modular inverses. Fermat's Little Theorem provides a convenient way to compute the modular inverse $a^{-1} \pmod p$ for any $a \in \mathbf{Z}_p^*$, where p is prime:

$$a^{-1} = a^{p-2} \pmod p \ .$$

(Euclid's extended algorithm for computing $\gcd(a, p)$ is more efficient.)

Primality testing. The converse of Fermat's Little Theorem is "almost" true. The converse would say that if $1 \le a < p$ and $a^{p-1} = 1 \pmod p$, then p is prime. Suppose that p is a large randomly chosen integer, and that a is a randomly chosen integer such that $1 \le a < p$. Then if $a^{p-1} \ne 1 \pmod p$, then p is certainly not prime (by FLT), and otherwise p is "likely" to be prime. FLT thus provides a heuristic test for primality for randomly chosen p; refinements of this approach yield tests effective for all p.

Exercise 1 *Prove that $\lambda(n)$ is always a divisor of $\phi(n)$, and characterize exactly when it is a proper divisor.*

Exercise 2 *Suppose $a > 1$ is not even or divisible by 5; show that a^{100} (in decimal) ends in `001`.*

Exercise 3 *Let p be prime. (a) Show that $a^p = a \pmod p$ for any $a \in \mathbf{Z}_p$. (b) Argue that $(a+b)^p = a^p + b^p \pmod p$ for any a,b in $\mathbf{Z}_p$. (c) Show that $(m^e)^d = m \pmod p$ for all $m \in \mathbf{Z}_p$ if $ed = 1 \pmod{p-1}$.*

Generators

Definition 1 *A finite group $G = (S, \cdot)$ may be* cyclic*, which means that it contains a* generator *g such that every group element $h \in S$ is a power $h = g^k$ of g for some $k \geq 0$. If the group operation is addition, we write this condition as $h = \underbrace{g + g + \cdots + g}_{k} = kg$.*

Example 1 *For example, 3 generates $\mathbf{Z}_{10}$ under addition, since the multiples of 3, modulo 10, are:*

$$3, 6, 9, 2, 5, 8, 1, 4, 7, 0 \; .$$

Fact 1 *The generators of $(\mathbf{Z}_m, +)$ are exactly those $\phi(m)$ integers $a \in \mathbf{Z}_m$ relatively prime to m.*

Example 2 *The generators of $(\mathbf{Z}_{10}, +)$ are $\{1, 3, 7, 9\}$.*

Example 3 *The group $(\mathbf{Z}^*_{11}, \cdot)$ is generated by $g = 2$, since the powers of 2 (modulo 11) are:*

$$2, 4, 8, 5, 10, 9, 7, 3, 6, 1 \; .$$

Fact 2 *Any cyclic group of size m is isomorphic to $(\mathbf{Z}_m, +)$. For example, $(\mathbf{Z}^*_{11}, \cdot) \leftrightarrow (\mathbf{Z}_{10}, +)$ via:*

$$2^x \pmod{11} \longleftrightarrow x \pmod{10} \; .$$

Theorem 1 *If p is prime, then $(Z^*_p, \cdot)$ is cyclic, and contains $\phi(p-1)$ generators. More generally, the group $(\mathbf{Z}_n, \cdot)$ is cyclic if and only if $n = 2$, $n = 4$, $n = p^e$, or $n = 2p^e$, where p is an odd prime and $e \geq 1$; in these cases the group contains $\phi(\phi(n))$ generators.*

Finding a generator of $\mathbf{Z}^*_p$. If the factorization of $p - 1$ is unknown, no efficient algorithm is known, but if $p - 1$ has known factorization, it is easy to find a generator. Generators of $\mathbf{Z}^*_p$ are relatively common ($\phi(n) \geq n/(6 \ln \ln n)$ for $n \geq 5$), so one can be found by searching at random for an element g whose order is $p - 1$. (Note g has order $p - 1$ if $g^{p-1} = 1 \pmod{p}$ but $g^{(p-1)/q} \neq 1 \pmod{p}$ for all prime divisors q of $p - 1$).

Group generated by an element. In any group G, the set $\langle g \rangle$ of elements generated by g is always a cyclic subgroup of G; if $\langle g \rangle = G$ then g is a generator of G.

Groups of prime order. If a group H has prime order, then every element except the identity is a generator. For example, the subgroup $QR_{11} = \{1, 4, 9, 5, 3\}$ of squares (quadratic residues) in $\mathbf{Z}^*_{11}$ has order 5, so 4, 9, 5, and 3 all generate QR_{11}. For this reason, it is sometimes of interest to work with the group QR_p of squares modulo p, where $p = 2q + 1$ and q is prime.

Exercise 1 *(a) Find all of the generators of $(\mathbf{Z}_{11}, \cdot)$ and of $(\mathbf{Z}_{2^k}, +)$. (b) Let g be a generator of $(\mathbf{Z}^*_p, \cdot)$; prove that g^x generates $\mathbf{Z}^*_p$ if and only if x generates $(\mathbf{Z}_{p-1}, +)$.*

Orders of Elements

Definition 1 *The* order *of an element a of a finite group G is the least positive t such that $a^t = 1$. (If the group is written additively, it is the least positive t such that $a + a + \cdots + a$ (t times) $= 0$.)*

	1	2	3	4	5	6	7	…	Order
1	**1**	1	1	1	1	1	1	…	1
2	**2**	**4**	**1**	2	4	1	2	…	3
3	**3**	**2**	**6**	**4**	**5**	**1**	3	…	6
4	**4**	**2**	**1**	4	2	1	4	…	3
5	**5**	**4**	**6**	**2**	**3**	**1**	5	…	6
6	**6**	**1**	6	1	6	1	6	…	2

Row a column k contains $a^k \bmod p$ for $p = 7$; boldface entries illustrate the fundamental period of $a^k \pmod p$ as k increases. The length of this period is the *order* of a, modulo p. By Fermat's Little Theorem the order always divides $p - 1$; thus a^{p-1} is always 1 (see the column marked with an uparrow). Elements 3 and 5 have order $p - 1$, and so are *generators* of $\mathbf{Z}_7^*$. Element 6 is -1, modulo 7, and thus has order 2.

Fact 1 *The order of an element $a \in G$ is a divisor of the order of G. (The order $|G|$ of a group G is the number of elements it contains.) Therefore $a^{|G|} = 1$ in G . Thus when p is prime, the order of an element $a \in \mathbf{Z}_p^*$ is a divisor of $|\mathbf{Z}_p^*| = p - 1$, and in general the order of an element $a \in \mathbf{Z}_n^*$ is a divisor of $|\mathbf{Z}_n^*| = \phi(n)$.*

Computing the order t of an element $a \in G$. If the factorization of $|G|$ is unknown, no efficient algorithm is known, but if $|G|$ has known factorization $|G| = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, it is easy. Basically, compute the order t as $t = p_1^{f_1} p_2^{f_2} \cdots p_k^{f_k}$ where each f_i is initially e_i, then each f_i is decreased in turn as much as possible (but not below zero) while keeping $a^t = 1$ in G.

Fact 2 *When p is prime, the number of elements in $\mathbf{Z}_p^*$ of order d, where $d \mid (p - 1)$, is $\phi(d)$. For example, since $\phi(2) = 1$, there is a unique square root of 1 modulo p, other than 1 itself (it is $-1 = p - 1 \pmod p$).*

Exercise 1 *Let $\mathrm{ord}(a)$ denote the order of $a \in G$. (a) Prove that $\mathrm{ord}(a) = \mathrm{ord}(a^{-1})$ and $\mathrm{ord}(a^k) \mid \mathrm{ord}(a)$. (b) Prove that $\mathrm{ord}(ab)$ is a divisor of $\mathrm{lcm}(\mathrm{ord}(a), \mathrm{ord}(b))$, and show that it may be a proper divisor. (c) Show that $\mathrm{ord}(ab) = \mathrm{ord}(a)\,\mathrm{ord}(b)$ if $\gcd(\mathrm{ord}(a), \mathrm{ord}(b)) = 1$.*

Exercise 2 *Show that there are at least as many elements of order $p - 1$ (i.e. generators) of $\mathbf{Z}_p^*$ as there are elements of any other order.*

Exercise 3 *Show that the order of a in $(\mathbf{Z}_n, +)$ is $n/\gcd(a, n)$.*

Euclid's Algorithm for Computing GCD

It is *easy* to compute $gcd(a, b)$. This is surprising because you might think that in order to compute $\gcd(a, b)$ you would need to figure out their divisors, i.e. solve the factoring problem. But, as you will see, we don't need to figure out the divisors of a and b to find their gcd.

Euclid (*circa* 300 B.C.) showed how to compute $\gcd(a, b)$ for $a \geq 0$ and $b \geq 0$:

$$\gcd(a, b) = \begin{cases} a & \text{if } b = 0 \\ \gcd(b, a \bmod b) & \text{otherwise} \end{cases}$$

The recursion terminates since $(a \bmod b) < b$; the second argument strictly decreases with each call. An equivalent non-recursive version sets $a_0 = a$, $a_1 = b$, and then computes a_{i+1} for $i = 2, 3, \ldots$ as $a_{i+1} = a_{i-1} \bmod a_i$ until $a_{i+1} = 0$, then returns a_i.

Example 1 *Euclid's Algorithm finds the greatest common divisor of 12 and 33 as:*

$$\gcd(12, 33) = \gcd(33, 12) = \gcd(12, 9) = \gcd(9, 3) = \gcd(3, 0) = 3 \, .$$

The equivalent non-recursive version has $a_0 = 12$, $a_1 = 33$, and

$$\begin{aligned} a_2 &= a_0 \bmod a_1 = 12 \bmod 33 = 12 \\ a_3 &= a_1 \bmod a_2 = 33 \bmod 12 = 9 \\ a_4 &= a_2 \bmod a_3 = 12 \bmod 9 = 3 \\ a_5 &= a_3 \bmod a_4 = 9 \bmod 3 = 0 \end{aligned}$$

So $\gcd(12, 33) = 3$.

It can be shown that the number of recursive calls is $O(\log b)$; the worst-case input is a pair of consecutive Fibonacci numbers. Euclid's algorithm (even if extended) takes $O(k^2)$ bit operations when inputs a and b have at most k bits; see Bach and Shallit.

Euclid's Extended Algorithm

Theorem 1 *For all integers a, b, one can efficiently compute integers x and y such that*

$$\gcd(a, b) = ax + by \ .$$

We give a "proof by example," using Euclid's Extended Algorithm on inputs $a = 9$, $b = 31$, which for each a_i of the nonrecursive version of Euclid's algorithm finds an x_i and y_i such that $a_i = ax_i + by_i$:

$$\begin{array}{lllllll}
a_0 & = & a & = & 9 & & & = & a*1+b*0 \\
a_1 & = & b & = & 31 & & & = & a*0+b*1 \\
a_2 & = & a_0 \text{ mod } a_1 & = & 9 & = & (a*1+b*0)-0*(a*0+b*1) & = & a*1+b*0 \\
a_3 & = & a_1 \text{ mod } a_2 & = & 4 & = & (a*0+b*1)-3*(a*1+b*0) & = & a*(-3)+b*1 \\
a_4 & = & a_2 \text{ mod } a_3 & = & 1 & = & (a*1+b*0)-2*(a*(-3)+b*1) & = & a*7+b*(-2) \\
a_5 & = & a_3 \text{ mod } a_4 & = & 0 & & & &
\end{array}$$

Thus Euclid's Extended Algorithm computes $x = 7$ and $y = -2$ for $a = 9$ and $b = 31$.

Corollary 1 (Multiplicative inverse computation) *Given integers n and a where $\gcd(a, n) = 1$, using Euclid's Extended Algorithm to find x and y such that $ax + ny = 1$ finds an x such that $ax \equiv 1 \pmod{n}$; such an x is the* multiplicative inverse *of a modulo n: $x = a^{-1} \pmod{n}$.*

Example 1 *The multiplicative inverse of 9, modulo 31, is 7. Check: $9 * 7 = 63 = 1 \pmod{31}$.*

Exercise 1 *Find the multiplicative inverse of 11 modulo 41.*

Exercise 2 *Prove that if $\gcd(a, n) > 1$, then the multiplicative inverse $a^{-1} \pmod{n}$ does not exist.*

Exercise 3 *Show that Euclid's algorithm is correct by arguing that d is a common divisor of a and b if and only if d is a common divisor of b and $(a \text{ mod } b)$.*

Chinese Remainder Theorem

When working modulo a *composite* modulus n, the Chinese Remainder Theorem (CRT) can both speed computation modulo n and facilitate reasoning about the properties of arithmetic modulo n.

Theorem 1 (Chinese Remainder Theorem (CRT)) *Let $n = n_1 n_2 \cdots n_k$ be the product of k integers n_i that are pairwise relatively prime. The mapping*

$$f(a) = (a_1, \ldots, a_k) = (a \bmod n_1, \ldots, a \bmod n_k)$$

is an isomorphism from $\mathbf{Z}_n$ to $\mathbf{Z}_{n_1} \times \cdots \times \mathbf{Z}_{n_k}$: if $f(a) = (a_1, \ldots, a_k)$ and $f(b) = (b_1, \ldots, b_k)$, then

$$\begin{aligned}
f((a \pm b) \bmod n) &= ((a_1 \pm b_1) \bmod n_1, \ldots, (a_k \pm b_k) \bmod n_k) \\
f((ab) \bmod n) &= ((a_1 b_1) \bmod n_1, \ldots, (a_k b_k) \bmod n_k) \\
f(a^{-1} \bmod n) &= (a_1^{-1} \bmod n_1, \ldots, a_k^{-1} \bmod n_k) \text{ if } a^{-1} \pmod{n} \text{ exists} \\
f^{-1}((a_1, \ldots, a_k)) &= a = \sum_i a_i c_i \pmod{n} \text{ where } m_i = n/n_i \text{ and } c_i = m_i (m_i^{-1} \bmod n_i)\ .
\end{aligned}$$

When $n = pq$ is the product of two primes, working modulo n is equivalent to working independently on each component of its CRT (i.e. (mod p, mod q)) representation. It can be worthwhile to convert an input to its CRT representation, compute in that representation, and then convert back.

Example: For $n = 35 = 5 \cdot 7$ put $(a \bmod 35)$ in row $a_1 = (a \bmod 5)$ and column $a_2 = (a \bmod 7)$:

	0	1	2	3	4	5	6
0	0	15	30	10	25	5	20
1	21	1	16	31	11	26	6
2	7	22	2	17	32	12	27
3	28	8	23	3	18	33	13
4	14	29	9	24	4	19	34

$$\begin{aligned}
f(8) &= (3, 1) \\
f(-8) = f(27) &= (-3, -1) = (2, 6) \\
f(12) &= (2, 5) \\
f(12^{-1}) &= (2^{-1}, 5^{-1}) = (3, 3) = f(3) \\
f(8 + 12) = f(20) &= (3 + 2, 1 + 5) = (0, 6) \\
f(8 \cdot 12) = f(96) = f(26) &= (3 \cdot 2, 1 \cdot 5) = (1, 5)
\end{aligned}$$

Here $m_1 = 7$, $m_2 = 5$, $c_1 = 7 \cdot (7^{-1} \bmod 5) = 7 \cdot 3 = 21$, $c_2 = 5 \cdot (5^{-1} \bmod 7) = 5 \cdot 3 = 15$, so

$$f^{-1}((a_1, a_2)) = 21 a_1 + 15 a_2 \pmod{35}\ .$$

(Note: $f(21) = (1, 0)$, $f(15) = (0, 1)$.) Thus, $f^{-1}((1, 5)) = 21 + 5 \cdot 15 = 96 = 26 \pmod{35}$.

Speeding up Modular Exponentation. A significant application is speeding up exponentiation modulo $n = pq$ when p and q are known. To compute $y = x^d \bmod n$, where $f(x) = (x_1, x_2)$:

$$f(y) = f(x^d) = (x_1^d \bmod p, x_2^d \bmod q) = (x_1^{d \bmod (p-1)} \bmod p, x_2^{d \bmod (q-1)} \bmod q)\ .$$

Note $x_1^{p-1} = 1 \bmod p$ for $x_1 \neq 0$ by Fermat's Little Theorem. Then convert back from $(y \bmod p, y \bmod q)$ to $y \bmod n$. Since exponentiation takes time cubic in the input size, two half-size exponentiations are about four times faster than one full-size exponentiation (including conversion).

Exercise 1 *Prove that x is a square mod $n = pq$ if and only if it is a square mod p and mod q.*